Ministry of Education of Republic of Belarus
BELARUSIAN STATE UNIVERSITY
RESEARCH INSTITUTE FOR APPLIED PROBLEMS
OF MATHEMATICS AND INFORMATICS

UDK 004.056.55
Registration No. 20181085

APPROVED
Director of RI APMI
Prof.

Yu. Kharin
«___» _____ 2018

R&D TECHNICAL REPORT

Security evaluation of the MAM (Masked Authenticated Messaging) cryptographic protocol

(final)

№ 84/2

Scientific supervisor
Head of IT Security reseach laboratory, PhD S. Agievich

Minsk 2018

Team List

Scientific supervisor

Head of a research laboratory, PhD	S. Agievich (Introduction, Chapter 1, Sections 2.9, 3.5, 4.1 — 4.3, 5.1, 5.2, 5.6, 5.8, Conclusion, Specification)

Executors

Scientific assistant	M. Kazlovsky (Sections 4.2, 4.4, Implementation [testing])
Junior scientific researcher	M. Koipish (Sections 3.1, 3.2, 3.4, 5.3, Subsections 5.2.8, 5.2.9)
Junior scientific researcher	K. Pavlov (Sections 3.3, 3.4, 5.5, Subsections 5.7.1, 5.7.4, Implementation [tree-traversal])
Scientific researcher	V. Semenov (Sections 2.1 — 2.5, 2.7, Implementation)
Leading scientific researcher, PhD	O. Solovey (Sections 2.1 — 2.4, 2.6, 2.8, 5.4, Subsections 5.7.2, 5.7.3)

Contents

Introduction

The IOTA initiative is devoted to the creation of an infrastructure for interaction between Internet-of-Things devices (IoT), hereinafter referred to as entities. One of the mechanisms for protecting messages transmitted between the entities is the Masked Authenticated Messaging (MAM) protocol. Message protection includes the following:

- message authentication by means of Digital Signature (DS) algorithms;
- Stream Encryption (SE) of messages;
- additional message authentication using Message Authentication Codes (MAC).

In MAM, the SE and MAC algorithms are applied together with the same secret key. In fact, SE and MAC are combined into a single mechanism usually called Authenticated Encryption (AE).

A secret key of AE is preliminarily transmitted between the entities (beyond the scope of MAM). Such a preliminarily transmission of secret keys is usually attributed as the PSK (Pre-Shared Key) key delivery mechanism.

DS algorithms use pairs of private and public keys. A key pair belongs to a node that wants to broadcast messages. This node stores its private key in secret and distributes the corresponding public key to the recipient entity. During the distribution, the public key is protected by MAC.

DS, SE and MAC algorithms are built upon a special cryptographic sponge function. The same function is also used to organize preliminarily hashing of messages before their processing by the DS algorithms. The sponge function is not detailed here, it is considered as an ideal cryptographic primitive.

The current version of MAM has the following drawbacks:

- only the PSK key delivery mechanism based upon secret keys is supported. Entities of MAM cannot communicate privately using more flexible mechanisms based upon public keys. To launch these mechanisms, MAM has to be extended by the PKE (Public Key Encryption) algorithms;
- the format of MAM messages is very strict: it doesn't allow the usage of several key delivery methods or the splitting of a long message into a sequence of short related fragments.

To overcome the drawbacks listed above, an enhanced version of MAM should be developed. Hereinafter, the extended MAM protocol is called MAM2.

The basic (initial) MAM2 protocol provides message splitting. The extended (final) MAM2 protocol provides the additional PKE delivery mechanism.

R&D “Security evaluation of the MAM (Masked Authentication Messaging) cryptographic protocol” is carried out on the basis of the contract No. 84/2.

The purpose of the R&D is the security evaluation of the MAM protocol, including its basic cryptographic primitives, and the designing of the MAM2 protocol.

Tasks of the R&D:

1. Security evaluation of the SE algorithms (Sections 2.2, 2.6).
2. Security evaluation of the MAC algorithm (Sections 2.2, 2.6).
3. Security evaluation of the coincidence of SE and MAC keys in AE (Subsection 2.1).
4. Security evaluation of the DS algorithms (Section 3).
5. Designing the format of MAM2 messages (Chapter 4).
6. Development of the basic MAM2 specification (Specification).
7. Developement of the PKE algorithms (Chapter 5).
8. Designing the binding between IOTA addresses and MAM public keys (Section 5.8).
9. Development of the extended MAM2 specification.
10. Implementation of the basic MAM2 specification.
11. Implementation of the extended MAM2 specification.

This technical report contains the results of processing these tasks.

Integral parts of this report are Specification and Implementation.

Chapter 1

Preliminaries

1.1 IOTA

IOTA is both a runtime environment and a transport layer of MAM2. MAM2 should comply with IOTA, should exploit its better sides and overcome its limits. In this chapter we provide a short overview of IOTA touching topics which are relevant for the further discussion of MAM2.

We should mention that there is a large lack of technical documentation about IOTA. We attribute this fact to the fast growth of the IOTA infrastructure during which developers are oriented mainly on writing programs, not reports. In result, we could understand some facts only from programs or from discussions with the IOTA community.

This is why this section is also an attempt of reconstructing IOTA concepts, it is a map which helps us to navigate between IOTA "islands and reefs". We have been writing this chapter incrementally while discovering new facts and digging into new layers.

1.2 Ternary logic

1.2.1 Motivation

Let b be a base of a numeral system. In usual (non-digital) life we generally use $b = 10$, in computer systems the choice $b = 2$ is occurring everywhere. But in the beginning of the computer era the choice of b was questionable. One line of arguments was about minimization of cost of storing data. Let us discuss this line.

To store integers from the set $\{0, 1, \ldots, N-1\}$, we should use approximately $\log_b N = \ln N / \ln b$ digits in base b. Suppose that the cost of storing of each digit is proportional to b. To optimize the total cost $b \ln N / \ln b$, we have to use b which minimizes the function $f(x) = x / \ln x$. The minimum of $f(x)$ is attained at $x = e = 2.71828$. Since b must be integer, we have to choose between $b = 2$ and $b = 3$. It could be easily checked that $b = 3$ (ternary logic) is the best choice.

IOTA is oriented on low-cost devices with the minimum power consumption. So in theory the choice $b = 3$ is best suited for IOTA. But in practice ternary hardware devices do not exist or at least are extremely rare, they have to be emulated by conventional binary devices and the cost of such emulation prevailes over ternary benefits. The IOTA community hopes that the situation will change and ternary devices will appear in the nearest future.

It is interesting that computers built on ternary logic existed in the not so remote past. One example is the Soviet computer *Setun* (Сетунь) created in 1958.[1] It could do 5500 additions or 3000 multiplications per second. The storage device of Setun consisted of ternary ferritoid cells.

The paper [132] contains a short overview of Soviet efforts in the field of ternary devices. The author disclaims the mentioned low-cost storage argument for ternary logic and provides another line of arguments:

- very natural representation of signed integers;
- the sign of an integer is determined by the sign of its highest trit;
- a simple comparison of absolute values of ternary numbers;
- branching by sign in ternary devices takes two times less than in binary ones;
- truncation of a number is equivalent to correct rounding while the rounding methods used in binary devices do not provide this;
- in a 3-input ternary summator, a carry to the next position occurs in 8 cases out of 27, while in a binary summator — in 4 cases out of 8;
- the 3-level signal is more resistant to the noise in transmission lines. This means that error-correction codes for ternary information are simpler than binary ones.

We hope that these points are enough important and future ternary devices will be better than their binary counterparts.

1.2.2 Trits

Let us introduce some formalism related to ternary logic. Basic entities of information are *trits*: 0, 1, and -1. It will be convenient to denote -1 by $\overline{1}$. Moreover, let $\overline{a} = -a$ for an arbitrary integer a.[2]

There exists a bijection between ternary words of length n and integers from the interval $\left[-\frac{1}{2}(3^{n-1}-1), \frac{1}{2}(3^{n-1}-1)\right]$: a word $a = a_0 a_1 \dots a_{n-1}$, $a_i \in \{\overline{1}, 0, 1\}$, unambiguously represents the integer

$$[a] = \sum_{i=0}^{n-1} a_i 3^i.$$

We use here the little-endian convention: the first trit of a word is the lowest one. It is interesting that $\overline{[a]}$ is represented by the word $\overline{a} = \overline{a}_0 \overline{a}_1 \dots \overline{a}_{n-1}$.

If m is a positive integer and x is an arbitrary integer, then $x \bmod m$ is the unique $r \in \{0, 1, \dots, m-1\}$ such that $m \mid (x-r)$ (m divides $x-r$). The set $\{0, 1, \dots, m-1\}$, from which the residue r is chosen, seems to be the most natural if we deal mainly with nonnegative integers. But due to the ternary logic we have to deal with negative integers as well. Therefore, it would

[1] `https://en.wikipedia.org/wiki/Setun`.

[2] The chosen notation is easily supported by the LaTeX command `\overline`. Unfortunately, HTML doesn't contain a standard direct analogue of this command. So we also reserve for $-a$ the alternative notation $\not{a}$: the `\sout` command from the LaTeX package `ulem` or the `<del>` tag in HTML.

be more convenient to determine residues in slightly different way: if m is an odd positive integer and x is an arbitrary integer, then $x \bmods m$ is a unique $r \in \{-(m-1)/2, \ldots, (m-1)/2\}$ such that $m \mid (x - r)$. Such r is called *the least absolute residue*. For example, $5 \bmod 3 = 2$ but $5 \bmods 3 = \overline{1}$.

Given two ternary words $a_0a_1 \ldots a_{n-1}$ and $b_0b_1 \ldots b_{n-1}$, let us add and subtract them tritwise $\bmods 3$. Denote the corresponding operations by $\oplus$ and $\ominus$:

$$a_0a_1 \ldots a_{n-1} \oplus b_0b_1 \ldots b_{n-1} = c_0c_1 \ldots c_{n-1}, \quad c_i = (a_i + b_i) \bmods 3,$$

and

$$a_0a_1 \ldots a_{n-1} \ominus b_0b_1 \ldots b_{n-1} = d_0d_1 \ldots d_{n-1}, \quad d_i = (a_i - b_i) \bmods 3.$$

For example, $\overline{1}01 \oplus \overline{1}1\overline{1} = 110$, $\overline{1}01 \ominus \overline{1}1\overline{1} = 0\overline{1}\overline{1}$.

1.2.3 Strings of trits

Let us introduce some useful notations and operations over ternary strings.

$\bot$	an empty word (a special object or event in other contexts: an unused variable, an error, etc.);
T	$\{\overline{1}, 0, 1\}$, the ternary alphabet;
T^n	the set of all words of length n in T;
T^*	the set of all words of finite length in T (including the empty word $\bot$ of length 0);
$\lvert a \rvert$	the length of $a \in \mathsf{T}^*$;
T^{n*}	the set of all words from T^* whose lengths are multiplies of n;
α^n	for $\alpha \in \mathsf{T}$, the word of n instances of α;
a_i, $a[i]$	the i-th trit of $a \in \mathsf{T}^n$: $a = a_0a_1 \ldots a_{n-1} = a[0]a[1]...a[n-1]$;
$a[\ldots l)$	for $a \in \mathsf{T}^n$ and $l \leqslant n$, the subword $a[0]a[1]...a[l-1]$;
$a[l \ldots)$	for $a \in \mathsf{T}^n$ and $l \leqslant n$, the subword $a[l]a[l+1]...a[n-1]$;
$a[l_1 \ldots l_2)$	for $a \in \mathsf{T}^n$ and $0 \leqslant l_1 < l_2 \leqslant n$, the subword $a[l_1]a[l_1+1]...a[l_2-1]$;
$a \parallel b$	the concatenation of $a, b \in \mathsf{T}^*$.

1.2.4 Trytes

In IOTA, ternary words of length 3 are called *trytes*.[3] There exist 3^3 different trytes, we identify them with integers from the interval $[\overline{13}, 13]$. In IOTA, these integers are encoded by symbols of the alphabet $\{\texttt{9}, \texttt{A}, \ldots, \texttt{Z}\}$. The encoding rules are presented in Table 1.1 (accordingly with [83] and `https://laurencetennant.com/iota-tools/`).

[3] For comparison, in the mentioned Setun computer a tryte consists of 6 trits.

Table 1.1: Trytes

Tryte	Integer	Code	Tryte	Integer	Code	Tryte	Integer	Code
000	0	9	001	9	I	00$\overline{1}$	$\overline{9}$	R
100	1	A	101	10	J	10$\overline{1}$	$\overline{8}$	S
$\overline{1}$10	2	B	$\overline{1}$11	11	K	$\overline{1}\overline{1}$	$\overline{7}$	T
010	3	C	011	12	L	01$\overline{1}$	$\overline{6}$	U
110	4	D	111	13	M	11$\overline{1}$	$\overline{5}$	V
$\overline{1}\overline{1}$1	5	E	$\overline{1}\overline{1}\overline{1}$	$\overline{13}$	N	$\overline{1}\overline{1}$0	$\overline{4}$	W
0$\overline{1}$1	6	F	0$\overline{1}\overline{1}$	$\overline{12}$	O	0$\overline{1}$0	$\overline{3}$	X
1$\overline{1}$1	7	G	1$\overline{1}\overline{1}$	$\overline{11}$	P	1$\overline{1}$0	$\overline{2}$	Y
$\overline{1}$01	8	H	$\overline{1}$0$\overline{1}$	$\overline{10}$	Q	$\overline{1}$00	$\overline{1}$	Z

The operations $\oplus$, $\ominus$ over trytes are naturally transferred over their codes. For example, $\texttt{H} \oplus \texttt{T} = \texttt{D}$, $\texttt{H} \ominus \texttt{T} = \texttt{O}$.

1.2.5 Trints

A tryte contains even less information than a byte. To support rather large integers or to address rather large buffers, we have to use a more informative unit.

We propose to use *trints*, that is, triple trytes interpreted as integers from the interval $[-9841, 9841]$. In C programming language, the trint's analogue is `int`. That is why the name.

Trints can be represented by 16-bit words. A possible analogue of 32-bit words are *long trints*: pairs of trints. Long trints run over the interval

$$[-193710244, 193710244].$$

Here $193710244 \approx 2^{27.5}$.

1.3 Cryptography

1.3.1 Quantum-proof cryptography

One of the distinguishing features of IOTA is its security in the Post-Quantum Epoch. In this Epoch such cryptography pillars as IFC (Integer Factorization Crypto), FFC (Finite Field Crypto) or even ECC (Elliptic Curve Crypto) would be broken. It is because quantum computers can solve underlying computational problems, exponentially or subexponentially hard in the current Pre-Quantum Epoch, in polynomial time.

There exist several alternative cryptography platforms that preserve their security (or don't lose it so dramatically as the platforms mentioned above) in the new Epoch. The main such *quantum-proof* platforms are HBC (Hash-Based Crypto), CBC (Code-Based Crypto), LBC (Lattice-Based Crypto), MBC (Multivariate-polynomial-Based Crypto).

In the last few years, quantum-proof platforms and their representatives have been under intensive analysis from the cryptographic community. This is due to NIST Post-Quantum

Crypto Competition (NPQCC), which is being held by the US National Institute of Standards and Technology (NIST).

NPQCC is very interesting for us because we have plans to supply MAM2 with quantum-proof public key encryption (PKE) algorithms. It would be good if the chosen platform supported also digital signature (DS) algorithms. And it would be great if both PKE and DS algorithms could be run with the same keys. We provide short information about quantum-proof alternatives for PKE and DS in Table 1.2.

Table 1.2: The main quantum-proof cryptographic platforms

Platform	Pro	Contra
HBC	Fast DS. Very compact basic crypto. Clear and very stable security bounds.	Inherent one-timeness. Large signatures in the permanent and stateless settings. Doesn't provide PKE.
CBC	Reasonably fast PKE. Stable strength of McEliece (PKE).	Large keys. Relatively new, untested and inefficient DS.
LBC	Fast PKE and DS. Short keys and signatures. Relatively stable strength NTRU (PKE).	A lot of successful cryptanalytic attacks for previous versions of NTRU. Relatively recent and untested DS.
MBC	Reasonably fast. Short enough private keys and extremely short signatures.	Many proposed schemes have been broken (SFLASH). Relatively recent and untested PKE.

The author of these lines doesn't share the opinion that the Post-Quantum Epoch is on the threshold. I rather tend to a skeptical point of view expressed, for example, in [139].[4] But we can't ignore people's fears of the *crypto-apocalypse* that are fueled by the prevailing opinion of the cryptographic community.[5]

NPCQCC introduces 5 levels of security. Let us explain them briefly.

1. Security is comparable to or greater than a block cipher with a 128-bit key against an exhaustive key search (e.g. AES128).

2. Security is comparable to or greater than a 256-bit hash function against a collision search (e.g. SHA256).

3. Security is comparable to or greater than a block cipher with a 192-bit key against an exhaustive key search (e.g. AES192).

4. Security is comparable to or greater than a 384-bit hash function against a collision search (e.g. SHA384).

5. Security is comparable to or greater than a block cipher with a 256-bit key against an exhaustive key search (e.g. AES256).

[4] See also the recent article by M. Dyakonov: `https://spectrum.ieee.org/computing/hardware/the-case-against-quantum-computing`.

[5] For example, the prominent cryptographer D. Bernshtein have recently bet $2048 that the RSA-2048 challenge will be publicly factored by a quantum computer by 2033.

1.3.2 Hash-based cryptography

The main crypto engine of multi-user systems like IOTA is DS (Digital Signature) algorithms which provide authenticity of messages sent between entities. In case of IOTA, the messages are actually IOTA tokens. Alice, a sender of a token, signs it, Bob and other entities verify Alice's signature and approve the transfer of the token.

To build DS, IOTA uses the HBC platform. More precisely, IOTA tokens are signed using the WOTS (Winternitz One-Time Signature) scheme.

The WOTS scheme is very simple, it is built using only two hash functions: $h\colon S \to S$, $S = \mathbf{T}^l$, and $H\colon \mathbf{T}^* \to \{0, 1, \ldots, w-1\}^n$ (hereinafter in the scope of IOTA). The number w is a parameter of WOTS. In IOTA it is equal to 27.

The first function is used to generate n partial public and private keys (sk_i, pk_i), where sk_i are chosen at random from $\mathbf{T}^l$ and

$$pk_i = h^{w-1}(sk_i), \quad i = 1, 2, \ldots, n.$$

Here h^j is the composition of j instances of h (h^0 is the identity mapping). Alice publishes the full public key $pk = (pk_1, pk_2, \ldots, pk_n)$ and keeps the corresponding full private key $sk = (sk_1, sk_2, \ldots, sk_n)$ in secret.

A message $X \in \mathbf{T}^*$ to be signed is processed using the second hash function H that firstly compresses X cryptographically and secondly achieves the following

Property 1. If $H(X) \neq H(X')$ then neither of the relations $H(X) \geqslant H(X')$ nor $H(X') \geqslant H(X)$ hold. Here $a \geqslant b$ means that $a[i] \geqslant b[i]$ for all $i = 1, 2, \ldots, n$.

This property is usually achieved in WOTS by checksums. But IOTA has chosen another way called *normalization*. We will discuss it (along with other topics of the WOTS instantiation in IOTA) in Chapter 3.

After calculating $H(X)$, Alice finds $m_i = H(X)[i]$, then

$$s_i = h^{m_i}(sk_i), \quad i = 1, 2, \ldots, n,$$

and returns $s = (s_1, s_2, \ldots, s_n)$ as the resulting signature. To verify it Bob tests if

$$h^{w-1-m_i}(s_i) \stackrel{?}{=} pk_i, \quad i = 1, 2, \ldots, n.$$

The security of WOTS depends on the hardness of the following cryptanalytic hash-based problems. Let us formulate these problems for an arbitrary hash function $h\colon R \to S$ which can be either actual h (with $R = S = \mathbf{T}^l$) or H ($R = \mathbf{T}^*$, $S = \{0, 1, \ldots, w-1\}^n$).

1. `Inv`$[h]$: Given $Y \in S$ find $X \in R$ such that $h(X) = Y$.
2. `2Pre`$[h]$: Given $X' \in R$ find another $X \in R$ such that $h(X) = h(X')$.
3. `Coll`$[h]$: Find distinct $X, X' \in R$ such that $h(X) = h(X')$.

We will motivate these problems in Chapter 3.

An algorithm that solves one of the problems above takes the description of h and, optionally, some *instance*. It is either Y in `Inv`$[h]$ or X in `2Pre`$[h]$. Formally, the way of instance

generation is a part of the target problem. But in cryptographic literature this part of $\texttt{Inv}$ or $\texttt{2Pre}$ is usually avoided (see, for example, [147, chapter 9]).

In the most cases below, we suppose that instances are generated at random: $Y = h(X)$ with $X \in_R R$ in $\texttt{Inv}[h](Y)$ and $X \in_R R$ in $\texttt{2Pre}[h](X)$. Hereinafter, the notation $u \in_R A$ means that u is chosen at random (uniformly independently of other choices) from a set A. In § 3.5 we consider some modifications of $\texttt{Inv}$ and $\texttt{2Pre}$ in which other ways of instance generation are used.

Let $|R| \geqslant |S| = N$. It is well-known that if h is cryptographically strong, then $\texttt{Inv}[h]$ and $\texttt{2Pre}[h]$ can be solved only in average time $O(N)$ and $\texttt{Coll}[h]$ — in average time $O(N^{1/2})$. Probabilities and corresponding average times here are over random instances and random tapes of algorithms. We are talking about complexities in the Pre-Quantum Epoch. The Post-Quantum Epoch is discussed in Section 1.3.4.

It would be convenient to differentiate cryptographic strength depending on the problem:

- h is *one-way* (OW) if $\texttt{Inv}[h]$ is hard;
- h is *second preimage resistant* (SPR) if $\texttt{2Pre}[h]$ is hard;
- h is *collision resistant* (CR) if $\texttt{Coll}[h]$ is hard.

The very important point is that WOTS is a *one-time* signature scheme. Indeed, signing m_i, the part of $H(X)$, Alice reveals s_i and after that the adversary Mallory can easily sign $m_i + 1, m_i + 2, \ldots, w - 1$. Generally, Mallory can forge a signature of every X' such that $H(X') \neq H(X)$ and $H(X') \geqslant H(X)$.

It is interesting that Property 1 guarantees that such X' doesn't exist. So Alice can safely sign not one but two messages provided that Mallory cannot substitute two Alice's signed messages by his own after signing. In fact, WOTS is W2TS provided protection from such an adversary! In [50], the security of W2TS against general adversary is discussed.

But WOTS isn't W3TS. Indeed, after getting signatures of X and X' Mallory can easily sign messages X'' such that $H(X'') \geqslant \min(H(X), H(X'))$, where min is taken component-wise. Property 1 doesn't guarantee that such X'' doesn't exist.

Partial Alice's public keys pk_i are hashed, the resulting hash value $addr = h^*(pk_1 \parallel \ldots \parallel pk_n)$ is interpreted as her *address*. Here h^* is an additional hash function $\mathbf{T}^* \to \mathbf{T}^k$. The verification equation can be transformed into the following:

$$h^*\left(h^{w-1-m_1}(s_1) \parallel \ldots \parallel h^{w-1-m_n}(s_n)\right) \stackrel{?}{=} addr.$$

It is interesting that Alice and Bob (who owns partial public keys pk'_i and produces partial signatures s'_i) can create the collective or *multisig* address

$$Addr = h^*(pk_1 \parallel \ldots \parallel pk_n \parallel pk'_1 \parallel \ldots \parallel pk'_n)$$

involved in the multisig verification:

$$h^*\left(h^{w-1-m_1}(s_1) \parallel \ldots \parallel h^{w-1-m_n}(s_n) \parallel h^{w-1-m_1}(s'_1) \parallel \ldots \parallel h^{w-1-m_n}(s'_n)\right) \stackrel{?}{=} Addr.$$

We will discuss the multisig technique in Section 1.6.

Addresses are used in IOTA for identification of entities. Since addresses are one-time, entities can't set up a long-term strategy of their relations.

For example, suppose that Alice publishes her address for getting donations. Alice informs other nodes that she accepts donations up to a certain date. As soon as the date comes, Alice gathers all donations and transfers them from the donation address to some new one. Alice signs the transfer and Mallory, an adversary, can potentially recover the private key of the donation address from the signature. After that, a new donation from Bob who doesn't know about Alice's deadline could be stolen by Mallory. The donation, possibly intercepted, can be easily blocked by IOTA: it is only necessary to reject transactions in which some of the destination addresses is already used. But such a blocking is against Alice who wants to get donations even after the deadline.

1.3.3 Permutation-based cryptography

WOTS runs with $l = 243$, $w = 27$ and $n \in \{27, 54, 81\}$. The function $H(X)$ is derived from a conventional hash function H^*: $\mathbf{T}^* \to \mathbf{T}^{3n}$ by grouping triples of output trits into trytes, interpreting these trytes as integers and incrementing these integers by 13.

In the first version of IOTA, a hash function named Curl was used both as h and H^* with only the difference that outputs of Curl-as-H^* are truncated up to $3n$ trytes.

Curl is built using the Permutation-Based Crypto (PBC) paradigm. It means that Curl uses a so-called *sponge function* F that maps from $\mathbf{T}^{729}$ to $\mathbf{T}^{729}$. A fragment of hashing data is written in a sponge state $S \in \mathbf{T}^{729}$, the state is transformed using F, the next and other fragments are processed in the same way. After processing (*absorbing* in the terminology of PBC) all the fragments, the first part of S is outputted (*squeezed*) as a resulting hash.

In fact, IOTA made an ambitious decision to base all its cryptography upon the sole permutation F. Unfortunately, in 2017 it was found that this permutation and therefore the resulting hash function Curl contain cryptographic weaknesses. After that IOTA initiated the development of a new hash function called Curl-P.[6] Later Curl-P was renamed to Troika ("triplet" in Russian). The specification of Troika was issued in December 2018 [179]. Troika follows the PBC paradigm and also uses a sponge function F with interface $\mathbf{T}^{729} \to \mathbf{T}^{729}$. It is a crucial point because F is used in IOTA not only for hashing but also for generating private keys from seeds. Moreover, F is extensively used in the AE part of MAM and are planning to be used in the future MAM2 (see Introduction and Chapter 2).

We discuss F in detail because right now IOTA uses a hash function that is incompatible with interface $\mathbf{T}^{729} \to \mathbf{T}^{729}$. This hash function, called Kerl, is again in the PBC class but its basic sponge function is binary: it transforms binary words of length 1600. The latter sponge function (Keccak-F) is a core of the SHA-3 hash family which was standardized in the US in 2015. In Kerl, trytes are transformed to bytes before absorbing and squeezed bytes are transformed back to trytes.

We believe that Kerl is a temporary solution and in the nearest future IOTA will use a sponge function F with the standard interface $\mathbf{T}^{729} \to \mathbf{T}^{729}$.

We will extensively use F in MAM2. We consider F cryptographically strong, as if it was chosen at random from the set of all permutations over $\mathbf{T}^{729}$.

[6] `https://blog.iota.org/iota-foundation-hires-cybercrypt-615d2df79001`.

1.3.4 Grover's algorithm

Let S be a finite set of cardinality N and f be a predicate $S \to \{0,1\}$. Let the equation $f(x) = 1$ have a unique solution in x. Grover's quantum algorithm [106] can find this solution in time $O(\sqrt{N})$ using $O(\sqrt{N})$ queries to an oracle which calculates $f(x)$.

Grover's algorithm is a search algorithm. More precisely, since an arbitrary (black-box) predicate f can be used, it is an unstructured search algorithm. It seems very surprising that we can find a unique solution of $f(x) = 1$ among N candidates calculating an unstructured f only $O(\sqrt{N})$ times. Grover explained this fact as follows: "*Quantum mechanical systems can be in a superposition of states and simultaneously examine multiple names*" (here *names* are different x). We can't provide further details due to a lack of expertise in the topic.

There exist several extensions of Grover's algorithm for solving hash-based problems. Let h be a target hash function from S to S.

1. The paper [45] provides variants of Grover's algorithm which solves $f(x) = 1$ even when the number M of solutions greater than 1 and even if M is unknown. These variants run in expected time $O(\sqrt{N/M})$.

   The modified Grover's algorithm can be used to solve the problems $\texttt{Inv}[h]$ and $\texttt{2Pre}[h]$. For example, if for a given x' we want to find $x \neq x'$ such that $h(x) = h(x')$, then we apply the algorithm with

   $$f(x) = \begin{cases} 1, & h(x) = h(x') \text{ and } x \neq x', \\ 0, & \text{otherwise.} \end{cases}$$

2. In [47] it was shown how to solve the problem $\texttt{Coll}[h]$ in time $O(N^{1/3})$ using $O(N^{1/3})$ queries to h. This is faster than the $O(\sqrt{N})$ time and queries taken by the pre-quantum "birthday"-type algorithms. The main idea is to precalculate $T = O(N^{1/3})$ values $h(x_1), \ldots, h(x_T)$, $x_t \in S$, and apply Grover's algorithm with the function

   $$f(x) = \begin{cases} 1, & h(x) \in \{h(x_1), \ldots, h(x_T)\} \text{ and } x \notin \{x_1, \ldots, x_T\}, \\ 0, & \text{otherwise.} \end{cases}$$

   The algorithm of [47] can be easily extended to h which maps from $R \supset S$ to S.

There exists some skepticism about Grover's algorithm and its extensions.

The first problem is that Grover's algorithm can't be parallelized at all. (It is an atonement for quantum speedup). All data units must be in one place and they must be processed simultaneously. For example, a quantum computer that solves $\texttt{Coll}[h]$ using the algorithm of [47] must process a giant precomputed list of $O(N^{1/3})$ values of f in full $O(N^{1/3})$ steps. For comparison, the pre-quantum ρ-algorithm is highly parallelizable. It can solve $\texttt{Coll}[h]$ using $O(N^{1/4})$ pre-quantum cores each of which makes $O(N^{1/4})$ steps. The cumulative classic time $O(N^{1/2})$ is greater than quantum one, but in fact small parallel classic cores outrun a sole giant quantum core finishing in time $O(N^{1/4})$ after the start.[7]

The second problem is that the quantum evaluation of f used in Grover's algorithm can be very cumbersome. We note that in the first example above each evaluation of f includes

[7] We should emphasize that the total amount of time of all small cores is still $O(N^{1/2})$.

the evaluation of h, where h means a cryptographic hash function of iterative nature with very inconvenient for quantum computers iterations.

In result, some researchers reject advantages of Grover's algorithm and its extensions over classic analogues. Some other researchers (see, for example, [28]) accept the first extension (inversion) and reject the second one (collision).[8]

The skepticism against Grover's algorithm is expressed in the materials of NIST: "*When all of these considerations are taken into account, it becomes quite likely that variants of Grover's algorithm will provide no advantage to an adversary wishing to perform a cryptanalytic attack that can be completed in a matter of years, or even decades.*" The irony is that these materials are devoted to hazards of quantum algorithms. From the beginning, NIST approves potential of some quantum algorithms and bans others.

We believe that we should be more consistent and assume the greatest potential of *all* quantum algorithms (all or nothing). Our final, maximum optimistic for Mallory and pessimistic for Alice, considerations are listed in Table 1.3. We remind that the target function $h\colon S \to S$ is assumed to be cryptographically strong, $|S| = N$, and in $\texttt{Coll}[h]$ the target function h can map from $R \supset S$.

Table 1.3: Complexity of hash-based problems

Epoch	Problem		
	$\texttt{Inv}[h]$	$\texttt{2Pre}[h]$	$\texttt{Coll}[h]$
Pre-Quantum	$O(N)$	$O(N)$	$O(N^{1/2})$
Post-Quantum	$O(N^{1/2})$	$O(N^{1/2})$	$O(N^{1/3})$

Entries of the table are in fact upper bounds on times of attacks against h. In many cases these upper bounds are also lower bounds, that is, we can replace O by Θ.[9] For example, in the recent paper [186] such a replacement is justified for the right bottom entry of the table provided that h is chosen at random.

Grover's algorithms can be applied not only to hash functions but also to other cryptographic constructions. For example,

- one of N keys of an encryption algorithm can be found in time $O(\sqrt{N})$ provided that an adversary knows enough lengthy plaintext and corresponding ciphertext;
- the internal collision in different sponge constructions can be found in time $O(N^{1/3})$ not $O(N^{1/2})$ provided that secret keys are not used. Here N is the total number of capacity parts of a sponge state (see Chapter 2 for details).

We should keep these facts in mind when designing AE algorithms.

1.4 Bundles and transactions

In IOTA, information about a transfer of coins (tokens) is encapsulated into a *bundle*. A bundle describes senders of coins, receivers of coins, amounts of coins. The bundle is hashed

[8] See also `https://blog.cr.yp.to/20171017-collisions.html`.

[9] Let us remind that $f(x) = \Theta(g(x))$ if $f(x) = O(g(x))$ and $g(x) = O(f(x))$.

and signed by all senders. Signatures are included in the bundle. There can be several senders and several receivers.

Possibly due to a rather large size of signatures, a bundle is divided into smaller parts called *transactions*. Transactions have a strict format described here: `https://docs.iota.org/introduction/iota-token/anatomy-of-a-transaction`.

The main parts of a transaction are:

- `sigMsgFragment` (or `sigMesssageFragment`) — a signature or its part. This field consists of 2187 trytes. Such number of trytes is sufficient for WOTS signatures of the 1st security level ($l = 81$). At the 2nd level ($l = 162$) signatures are presented by 2 parts, at the 3rd level ($l = 243$) — by 3 parts;
- `address` — the sender's or receiver's address;
- `value` — an amount of coins which is debited from the sender's account or credited to the receiver's account;
- `currentIndex` — the index of this transaction in the bundle;
- `lastIndex` — the total amount of transactions in the bundle minus 1;
- `bundle` — the hash of all the bundle data. This hash identifies the bundle. Exactly this hash is signed by senders;
- `trunkTransaction`, `branchTransaction` — references to other transactions (will be explained later);
- `nonce` — a volatile data that proves some computational work with this transaction (PoW, Proof of Work).

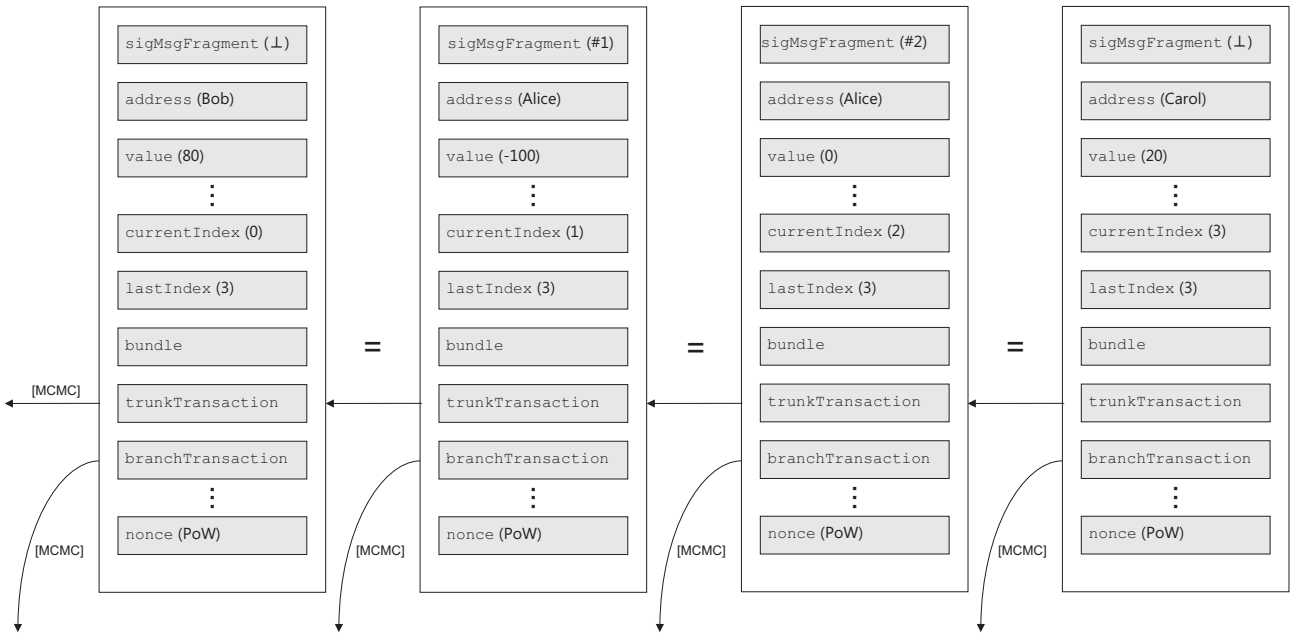

Figure 1.1: A typical bundle

Coins are transferred by the following rules.

1. Senders agree on a list of receivers and their coins.

2. Senders prepare a bundle and divide it into transactions.

3. Senders sign the bundle.

   The typical bundle describes a transfer from 1 sender to 2 receivers.[10] Such a bundle usually uses WOTS signatures of security level 2 ($l = 162$) and contains 4 transactions (see Figure 1.1). In the 1st and 4th transactions the `address` fields are addresses of the 1st and 2nd receivers (Bob and Carol) and the `value` fields are amounts of coins which will be credited to their accounts. The `sigMsgFragment` field is ignored. In the 2nd transaction, `address` is an address of a sender (Alice) and `value` is an amount of coins which will be debited from her account. This amount is presented with minus sign. The `sigMsgFragment` field contains the first part of Alice's signature. To simplify the overall coin processing logic, Alice has to withdraw all coins from her account. It is not a problem because Carol's address can be equal to Alice's address and therefore Alice = Carol will get a change. In the 3rd transaction, `address` is again the Alice's address, `value` is zero, `sigMsgFragment` contains the second part of the signature.

   Rules for multi-sender bundles are not strictly defined in IOTA documents (at least we couldn't find such documents). We guess that each sender is represented by one, two or three transactions with her address and signature or parts of the signature. The number of transactions is determined by the WOTS security level. As we indirectly understood, different senders can use different security levels.

   It is interesting that there exist proposals about multi-signatures.[11] It is the case when a receiver's address is a hash of concatenated addresses of several entities. To spend coins from this address all involved entities have to sign the corresponding transfer. Technically, it can be done rather easily if all target signatures are presented in sequential transactions of the bundle. In these transactions all `address` fields contain the multisig address, `value` of the first transaction contains the debited amount of coins, `value` of the next transactions are zero.

1.5 MAM

Let us stop here to discuss MAM (Masked Message Authentication). In a nutshell, MAM is a messaging system inside IOTA. MAM messages can be irrelevant to IOTA needs (see, for example, [48]), or MAM messages can serve these needs (can provide, for example, permanent channels for delivering one-time IOTA public keys / addresses, see [166]).

MAM messages are presented as special bundles. Transactions of such a bundle are fragments of the corresponding message. These fragments are written in the `sigMsgFragment` fields of the transactions. Let us remind that `sigMsgFragment` consists of 2187 trytes. It is

$$\frac{2187}{2673} = \frac{9}{11} = 0.8181\ldots$$

[10] `https://docs.iota.org/introduction/iota-token/bundle-construction`.

[11] `https://github.com/iotaledger/wiki/blob/master/multisigs.md`, `https://medium.com/@abmushi/iota-multisig-explained-bca334d250a2`.

of the entire transaction.

In each transaction of a MAM bundle the `value` field is zero and this property serves as an indicator of MAM[12] and blocks the interpretation of `sigMsgFragment` as WOTS signatures.

MAM messages are grouped into *channels*. Each channel has a unique identifier called `channelId`. This identifier is duplicated in the `address` field of each transaction of each channel's bundle. IOTA provides a command to fetch all transactions with a given `address`. This command can be used to get messages of the channel.

In MAM, the Merkle Signature Scheme (MSS) is used instead of WOTS. The main object of MSS is a Merkle tree which covers N standard WOTS public keys. The *root* of the tree is a permanent public key. As usual addresses, *root* belongs to the set $\mathbf{T}^n$, $n = 243$. The identifier `channelId` is either *root* or $h(root)$ or $h(root \parallel \mathrm{PSK})$, where PSK is a pre-shared key which is delivered between entities outside IOTA.

Alice signs every MAM message. To do this Alice uses a fresh WOTS private key in the standard WOTS signing algorithm and continues the resulting WOTS signature with the path in the Merkle tree from the corresponding one-time public key to *root*. This path consists of approximately $\log_2 N$ elements of $\mathbf{T}^n$.

Every MAM message is protected using AE algorithms. A key K is either *root* or *root* $\parallel$ PSK. There are 3 choices for `channelId` and 2 choices for K. Only 3 combinations are allowed. These combinations relate to so-called MAM channel modes (see [48] for details):

- public: `channelId` $= K = root$;
- private: $K = root$, `channelId` $= h(K)$;
- restricted: $K = root \parallel \mathrm{PSK}$, `channelId` $= h(K)$.

It is interesting that MSS algorithms process a key-dependent hash value (in fact, a message-authentication-code, MAC) and that the MSS signature is encrypted along with the target MAM message.

As we see, MAM messages are divided into rather small fragments. And as we will see, MAM messages (as well as regular IOTA transactions) are processed by rather complicated rules. Therefore, IOTA transport for MAM/MAM2 messages is rather expensive and we should take this fact into account when designing MAM2.

1.6 The Tangle

Let us continue the discussion how the IOTA transfer is processed.

4. Senders send the signed bundle to the Tangle.

   Here the *Tangle* is a database which contains all the IOTA transactions and in which all transactions reference each other. More precisely, each new transaction contains references to two different previous transactions. We can imagine directed graph which nodes are transactions and which edges are references. Each node except the first one (it is called *genesis*) has out-degree 2. The graph is acyclic because older transactions can't reference newer ones.

[12] It is only our guess. It should be approved (or disapproved) by the IOTA community.

The Tangle is managed by an entity called *Coordinator*. The existence of Coordinator contradicts decentralization rules that are usual in the cryptocurrencies sphere. IOTA has plans to eliminate Coordinator in the nearest future. But Coordinator still exists, so let us proceed with the current settings. Other plans are to divide the Tangle into subtangles. We again ignore these plans in this report.

5. Senders:

   1) for each transaction determines its references to other transactions in the Tangle and writes these references in the `trunkTransaction` and `branchTransaction` fields. Senders use a special Markov-Chain-Monte-Carlo (MCMC) algorithm. Senders do not apply MCMC for the `branchTransaction` fields in the second and further transactions of the bundle. These fields reference to exactly previous transactions of the bundle;

   2) for each bundle's transaction make some computations and write the corresponding PoW in the `nonce` field.

   It is not clear how senders perform these steps jointly. We suppose that there exists a mechanism for choosing one of the senders for MCMC + PoW. Interestingly, the chosen sender should be full. It means that the sender stores (or has access to) a sufficiently large part of the Tangle to get adequate MCMC results. As we understand, in earlier versions of IOTA, senders can delegate MCMC + PoW to Coordinator.

6. Coordinator:

   1) verifies all bundle's transactions (format + signatures). The verification is not performed if Coordinator understands that a MAM bundle is processed;[13]

   2) writes tailored transactions in the Tangle.

Let us note that Coordinator doesn't sign either transactions or the whole bundle. The main protection tool is PoW. It could be a problem because IOTA announced the possibility of very small PoW which can be created by lightweight IoT entities. If Coordinator is eliminated then adversaries could easily produce many "different futures" of the Tangle at feasible times. Such and other issues about the dynamics of the Tangle are treated in the white paper [164] from the probabilistic point of view.[14]

Right now, the consistency of the Tangle is sustained by an additional mechanism described below. Time to time, Coordinator issues special bundles called *milestones*. A milestone does not transfer useful information, they are just "trust anchors" which approve recent parts of the Tangle. As far as we understand, MCMC takes into account these anchors in its work.

Accordingly to [78], an usual milestone consists of two transactions. In the first (head) transaction, the `trunkTransaction` and `branchTransaction` links are configured in the usual way. In the second (tail) transaction, `trunkTransaction` refers to the first transaction, and `branchTransaction` repeats its `trunkTransaction` link. Such transaction settings make milestone bandles different from regular ones.

[13] An unanswered question to the IOTA community: Is it true that through MAM messages everyone can put everything in the Tangle?

[14] See also `https://www.iota.org/research/academic-papers`.

Milestone bundles are signed using MSS, not WOTS. Indeed, Coordinator's public key should be enough permanent, so WOTS is not suitable. The height of the Merkle tree used is 20, so Coordinator can sign 2^{20} milestone bundles. It is unclear how the transition to Coordinator's new public key (root key) will be made. Right now, this root key is stored in a special config file of client software. It looks like a root certificate of conventional PKIs.

The Tangle stores all IOTA transactions. Using the Tangle, everybody can understand how many coins belong to the entity with a particular address or, in other words, what is the *balance* of the address. In IOTA, the list of pairs (address, balance) is called a *snapshot*. (Its synonym, used, for example, in Bitcoin, is UTXO: Unspent Transaction Output).

Potentially, each entity (for example, a newcomer) can download all the Tangle, that is, all history of IOTA. But in many cases it is not necessary or even impossible: snapshots or even partial snapshots can be used instead of the Tangle.

There exist a special API, called IRI, to communicate with Coordinator. The documentation can be found here: `https://iota.readme.io/reference`. Using IRI commands, entities can, for example, get snapshots for given addresses (`getBalances`) or find transactions by address, bundle or even referring transactions (`findTransactions`).

IRI can be used to attach MAM messages to the Tangle and fetch from it. It is necessary to run `findTransactions` with `channelId` as `address`.

1.7 MAM2 expectations

We present our understanding of future MAM2 features. We do it in the form of questions and answers. The answers given are preliminary, they can be (and should be) discussed and adjusted.

1. Should MAM2 support broadcast (one-to-many) messages?

   Yes. We think that broadcasting is the main purpose of MAM2. We should follow the line introduced in MAM when Alice opens a *channel* through which she broadcasts series of messages to Bob, Carol and other entities (see [48] for details).

   The only addendum to MAM that seems important to do is the partitioning of a message into *packets* (fragments). In total, we can imagine the following picture: MAM2 is a tuple of channels, a channel is a tuple of messages, a message is a sequence of packets.

2. Should MAM2 support peer-to-peer (end-to-end) messages?

   Yes. That is why we are going to develop PKE (Public Key Encryption) in Phase 2 of our project. The easiest way to organize broadcasting is to use a pre-shared key (PSK) common to all members of the broadcasting group.

3. Should MAM2 support sending a message simultaneously to several broadcasting groups and several PKE-peers?

   Maybe. Such functionality seems attractive but its implementation can greatly complicate the protocol.

4. Should MAM2 support multiple senders?

No. This functionality is very difficult to support. The main problem is preparing a message before sending it. If there are several senders then all of them have to approve the message. Such approving requires preliminary interactions between senders. In fact, to send a message they have to send each other a lot of preliminary messages.

5. Should sender's addresses be permanent?

   Yes. One of the purposes of MAM is to make a permanent link to one-time IOTA addresses. MAM2 should follow this line.

6. Should MAM2 support detached messages?

   Yes. The Tangle can only contain information about a sender and his/her keys for a message. The payload parts of messages can be taken out of the Tangle.

7. Should MAM2 support various options for securing messages?

   Yes. A sender can turn on or turn off encryption or signing. Moreover, it should be possible to sign only certain fragments of a message.

Chapter 2

Authenticated Encryption Algorithms

2.1 General approaches to authenticated encryption

Confidentiality and authenticity are the two separate basic security goals. Encryption is the cryptographic primitive to achieve data confidentiality. But encryption does not directly guarantee message integrity or authenticity. The cryptographic primitive that guarantees that the data has not been modified is called Message Authentication Code (MAC). In turn, the cryptographic primitive whose goal is to provide confidentiality, integrity and authenticity of the encapsulated data, is called Authenticated Encryption (AE) or Authenticated Encryption with Associated Data (AEAD). The difference between AE and AEAD is that the latter provides authentication of public data (known as Additional Authenticated Data) which does not require confidentiality and is only authenticated and not encrypted. In this report the term "AE" refers to both categories unless explicitly stated otherwise.

In AE, an encryption algorithm applied by the sender takes a key, additional authenticated data (also referred as a header) and a plaintext. The algorithms returns a ciphertext. A decryption algorithm applied by a receiver takes the same key and additional authenticated data, a ciphertext and returns either a plaintext or a special symbol $\perp$ indicating that the additional authenticated data or/and ciphertext are invalid or unauthentic [23].

Further the message encryption and authentication algorithm $\overline{\mathcal{E}}$ is denoted by the term *wrapping*. In turn, the message decryption and verification algorithm $\overline{\mathcal{D}}$ is denoted by the term *unwrapping*.

There are several approaches to AE. A classical one is a general composition combining an encryption primitive and a MAC which usually employ separate keys. The encryption algorithm is required to be secure under a Chosen Plaintext Attack (CPA) and the MAC is required to be unforgeable under a Chosen Message Attack (CMA) [181].

Throughout this chapter, $\mathcal{E}$ and $\mathcal{D}$ denote encryption and decryption algorithms respectively and $\mathcal{T}$ denotes MAC. The algorithm $\mathcal{E}$ takes as input a secret key K_e and a plaintext message M and outputs a ciphertext C, i.e. $C = \mathcal{E}(K_e, M)$. The algorithm $\mathcal{D}$ takes as input a secret key K_e and a ciphertext C and outputs a plaintext message M, i.e. $M = \mathcal{D}(K_e, C)$. A MAC $\mathcal{T}$ takes as input a secret key K_m and a message M and outputs a tag T, which is usually shorter than M, i.e. $T = \mathcal{T}(K_m, M)$.

There are three widespread general compositions of an encryption and a MAC primitives [23]:

1. *Encrypt-and-MAC (EaM).* In order to wrap a message the plaintext is encrypted and authenticated separately:

$$\overline{\mathcal{E}}(K_e \parallel K_m, M) = (\mathcal{E}(K_e, M), \mathcal{T}(K_m, M)).$$

   In order to unwrap a message first the ciphertext is decrypted and then the authentication tag of the plaintext is verified.

2. *MAC-then-Encrypt (MtE).* In order to wrap a message first the plaintext is authenticated and then the plaintext together with the authentication tag is encrypted.

$$\overline{\mathcal{E}}(K_e \parallel K_m, M) = \mathcal{E}(K_e, M \parallel \mathcal{T}(K_m, M)).$$

   In order to unwrap a message first the ciphertext is decrypted and then the authentication tag of the plaintext is verified.

3. *Encrypt-then-MAC (EtM).* In order to wrap a message first the plaintext is encrypted and then the ciphertext is authenticated:

$$\overline{\mathcal{E}}(K_e \parallel K_m, M) = (C, \mathcal{T}(K_m, C)),$$

   where $C = \mathcal{E}(K_e, M)$. In order to unwrap a message first the authentication tag of the ciphertext is verified and then the ciphertext is decrypted.

The approach EtM has been shown [58] to be generically secure provided that the underlying primitives meet the appropriate security requirements. And in [131] it was proved that the EaM and MtE approaches are not generically secure (meaning not all secure encryption and MAC algorithms result in a secure wrapping algorithm). However, some implementations of EaM and MtE have been shown to be secure [181]. It is well-known that the EaM approach is the least secure among the three presented above (sending a tag weakens the security of the encryption scheme as it is leaking some information about the plaintext). This approach is not guaranteed to be secure for implementing secure channels [131].

Although the EtM approach is secure it is slow and requires two independent keys for encryption and authentication. More efficient AE schemes are dedicated schemes. Such schemes in particular are AE schemes based on a block cipher [125, 169]. Some of them, such as GCM [145] and CCM [82], are "two-pass modes", which simulate generic composition, except that they avoid the usage of two independent keys. Others, such as OCB [169] and XCBC [97], are "single-pass modes", which do not simulate generic composition and are, at least potentially, more efficient.

The alternative to block-cipher-based AE schemes and the topic of the current chapter are dedicated AE schemes based on the *sponge construction* (also referred as the sponge-based AE). The sponge construction is based on an ideal transformation or permutation with an appropriate selection of security parameters and can be used for building of provably secure cryptographic primitives [43].

2.2 General sponge constructions

2.2.1 The Sponge construction

The `Sponge` construction is a relatively new cryptographic primitive. It is a simple iterated construction for building a cryptographic function with variable-length input and arbitrary

output length based on fixed-length transformation or permutation F operating on $\mathbf{A}^b$ where $\mathbf{A}$ is a finite alphabet [30, 43]. Note, `Sponge` constructions are usually act on b bit state. For the purpose of generality we will refer here to $\mathbf{A}$ as the set of either bits or trits.

The state $S \in \mathbf{A}^b$ of the `Sponge` construction is split into two contiguous portions: the *outer state* $S_r \in \mathbf{A}^r$, which is accessible externally, and the *inner state* $S_c \in \mathbf{A}^c$, which is hidden. The value r is called the *rate* and the value c is called the *capacity*. The size $b = r + c$ of the entire state $S = S_r \parallel S_c$ is called *width*. Different values of rate and capacity give the trade-off between speed and security. The higher rate, the faster cryptographic functions and they are less secure.

The `Sponge` construction proceeds in two phases: the *absorbing phase* followed by the *squeezing phase* [31]. The input is absorbed in the first phase and the output is squeezed out in the second phase.

The output Z of the `Sponge` construction is given as

$$Z = \texttt{sponge}[F, \texttt{pad}, r](M, l),$$

where `pad` is a padding function for the input, r is the rate, M is the input (the message or the other data), and l is the desired output length [126]. The `Sponge` construction re-initializes its internal state between calls to it.

The `Sponge` construction works as follows. First, the state is initialized with a constant value (usually all zeros). The padding function `pad` is then applied to M to make it's length a multiple of r. In the absorbing phase the r-digit input message blocks are XORed into the r-digit outer state, interleaved with applications of the function F. When all message blocks are processed, the `Sponge` construction switches to the squeezing phase. Squeezing consists of returning the r-digit outer state as output blocks, interleaved with applications of the function F. The number of iterations is determined by the requested length l. Finally the output is truncated to an output string Z. The `Sponge` construction is illustrated in Figure 2.1.

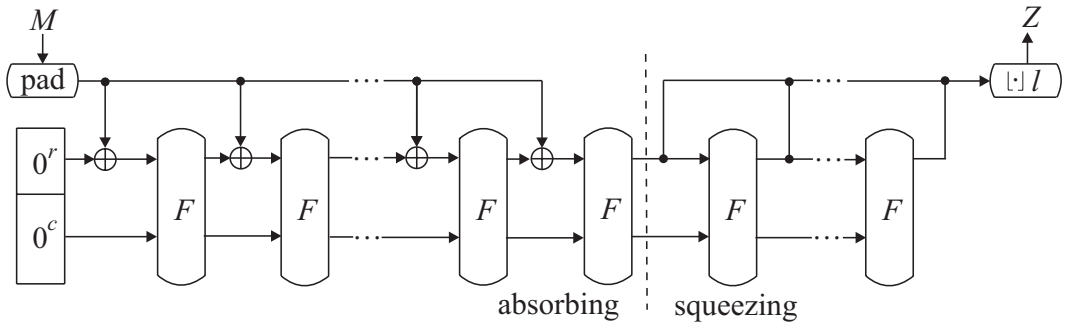

Figure 2.1: The `Sponge` construction

The security of the `Sponge` construction is based on the assumption that the underlying function F is secure. The `Sponge` construction, when used properly, is said to be secure against *generic attacks*. Generic attacks are attacks that do not exploit the properties of the concrete primitive but only the properties of the construction [30].

`Sponge` constructions can serve to implement various cryptographic primitives such as hash function, message authentification code (MAC), stream cipher, pseudorandom number generator (PRNG). For the possibility of using `Sponge` construction in a key-dependent cryptographic algorithms should be used the keyed sponge construction.

Bertoni et al. [33] introduced the keyed sponge construction as a simple evaluation of the `Sponge` construction over a concatenation of the key and the message, i.e. $\texttt{sponge}(K \parallel M)$. This type of construction is denoted as *outer-keyed*.

Chang et al. [62] considered a variant of the keyed sponge construction where the key is processed in the inner part of the `Sponge` construction. They observed that it can be seen as the `Sponge` construction based on an Even-Mansour construction [85]. This type of construction is denoted as *inner-keyed*.

2.2.2 The Duplex construction

The `Duplex` construction [31] is a construction closely related to the `Sponge` construction. Parameters for the `Duplex` construction are mostly the same as for the `Sponge` construction (uses function F, `pad` and r). The main differences are that the `Duplex` construction maintains its internal state between calls rather than re-initializing it and that there no longer exists a clear separation between the absorbing and squeezing phases [126]. The `Duplex` construction results in an object which accepts calls taking an input and returning an output which depends on all input received so far. Such an object, called a duplex object, is denoted as D. The call to a specific duplex object D is marked by its name D and dot.

A duplex object D has a state of b digits. Upon initialization all the digits of the state are set to zero. The function which processes inputs and produces outputs is called *duplexing*:

$$Z_i = D.\texttt{duplexing}(\sigma_i, l_i),$$

where σ_i is the i-th input and Z_i is i-th output, which is truncated to l_i digits. The maximum number of digits l_i is r and the input σ_i shall be short enough such that after padding it results in a single r-digit block. Inputs are absorbed and processed at the same time that outputs are squeezed (see Figure 2.2). For a duplex object it is possible to have an empty input (in which case it's still padded up to a full block) or to not produce an output (in which case the output is simply not used). Call where σ_i is the empty input are referred to as *blank call* while call with $l_i = 0$ are referred to as *mute call*.

Note, that padding has non-zero minimum length δ which effectively reduces the length of input σ to be $\rho = r - \delta$.

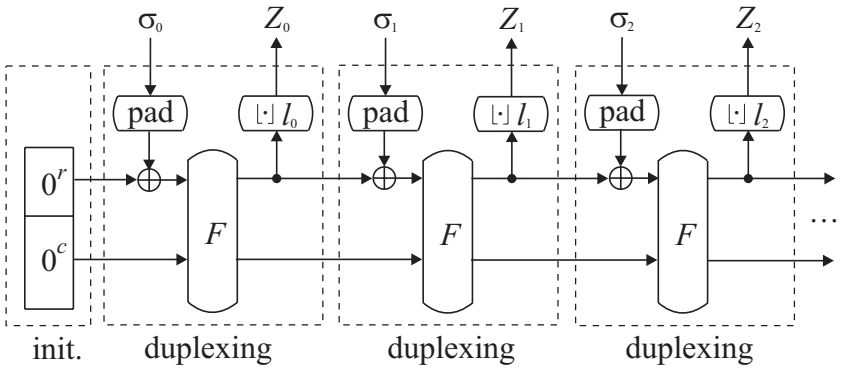

Figure 2.2: The `Duplex` construction

The `Duplex` construction is as secure as the `Sponge` construction with the same parameters [31].

2.2.3 The MonkeyDuplex construction

The `MonkeyDuplex` construction [34] is a modified version of the `Duplex` construction and meant to be used in a keyed mode. This construction is aimed at building stream ciphers and AE schemes. The `MonkeyDuplex` construction uses a permutation F with a tunable number of rounds. The performance of the construction can be optimized by reducing the number of rounds.

The `MonkeyDuplex` construction has the following parameters that determine its efficiency and security strength [34]:

- F : permutation family parameterized by the number of rounds;
- l_{key}: length of the key K;
- l_{nonce}: length of the nonce N;
- n_{init}: number of rounds applied after the key and nonce have been put in the state;
- n_{duplex}: number of rounds in a duplex call;
- r: rate during duplexing.

For achieving the required efficieny of construction and difficulty of state recovery one can vary parameters n_{duplex} and r where increasing the rate r necessitates increasing n_{duplex} and vice versa. Typically $n_{duplex} < n_{init}$. Furthermore, the `MonkeyDuplex` construction allows data encryption in stream mode with $n_{duplex} = 1$.

The b-digit state is initialized with the string $I = K \parallel N$, extended to b digits with padding (input I is unique for each use). Subsequently, n_{init} rounds of the permutation F are applied to the state. The duplexing calls are qualitatively the same as in the `Duplex` construction: after input σ_i are injected, n_{duplex} rounds of permutation F is applied to the state and the first $l_i \leqslant r$ digits of the state are extracted as output (see Figure 2.3). This output depends on all inputs received so far.

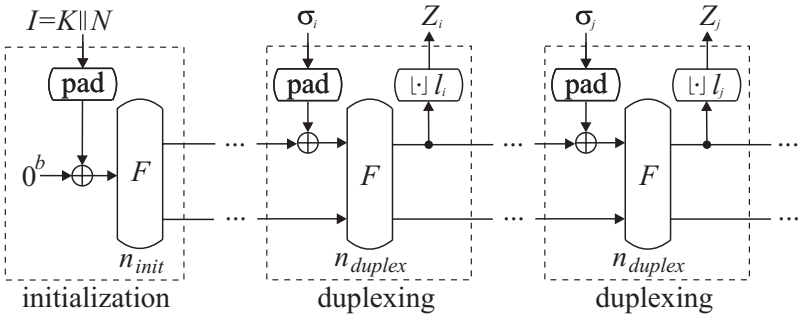

Figure 2.3: The `MonkeyDuplex` construction

2.2.4 The Full-State Keyed / Duplex Sponge constructions

Mennink et al. [148] proposed the full-state keyed sponge (FKS) and full-state keyed duplex (FKD) constructions. For these constructions the key is explicitly used to initialize the inner

state and the absorption phase is performed on the entire state. Squeezing, however, remains limited to the outer part of the state. Increasing the input block length allows to achieve higher efficiency in comparison with the classical constructions.

FKS in addition to the usual parameters of the `Sponge` construction has additional parameter $k \leqslant c$. FKS gets as input a key $K \in \mathbf{A}^k$, a message $M \in \mathbf{A}^*$, and a number $z \in \mathbb{N}$, and outputs a string $Z \in \mathbf{A}^z$. First, the state is initialized to all zeros with the key K added to the inner state. Next, the message M is padded and split into b-digit blocks. These blocks are sequentially absorbed with the function F. After absorption, r digits of the outer state are output and the state is processed via F until z output digits are obtained. The process of FKS is depicted in Figure 2.4.

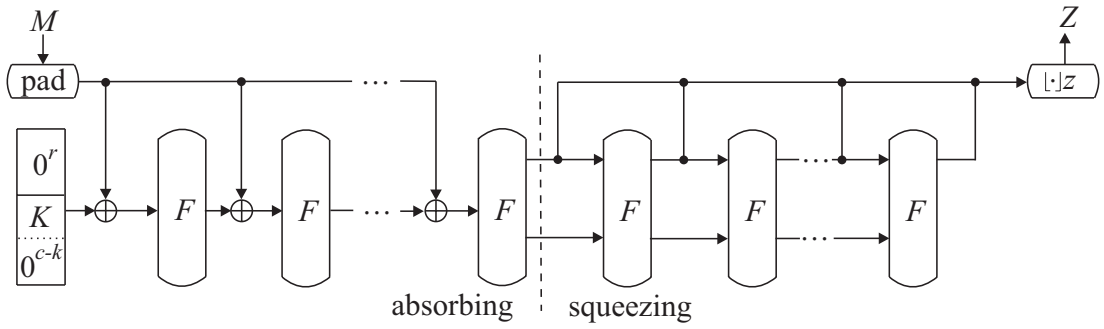

Figure 2.4: FKS construction

FKD is a generalization of the `Duplex` construction. The parameters of FKD are similar to FKS. FKD has two interfaces: initialization and duplexing. Upon initialization all the digits of the state are set to zero and the inner state is explicitly initialized with the key K. The message M is viewed as m sequential message blocks $M_0, M_0, \ldots M_{m-1}$. The block M_i shall be short enough such that after padding it results in a single b-digit block. The duplexing takes as input a message block M_i and a number $z_i \in \mathbb{N}$, where $z_i < r$, and it outputs a string $Z_i \in \mathbf{A}^{z_i}$. The process of FKD is depicted in Figure 2.5.

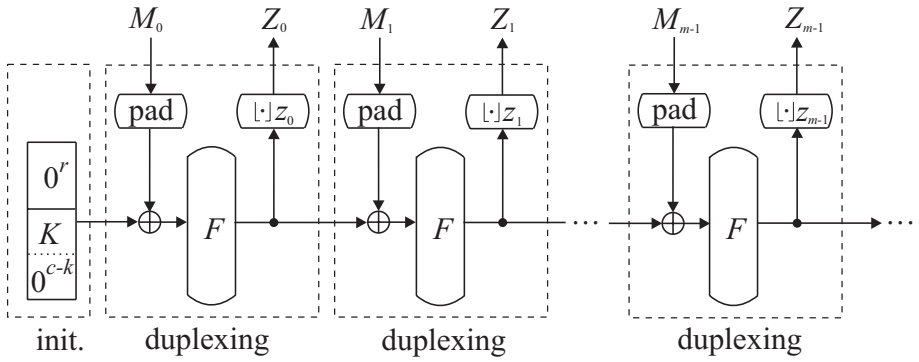

Figure 2.5: FKD construction

The generic security of the FKS and FKD is proved to be quite close to that of the classic keyed sponge and `Duplex` construction [148].

2.2.5 Internal mechanisms

The `overwrite` mode. This mode was introduced in [31]. It is a variant of the `Sponge` construction which absorbs data by overwriting the outer state with the input rather than XORing it in. This has a "forgetting" effect once overwritten outer state is discarded. Overwriting has the advantage over XORing that between calls to F only outer state must be kept instead of full state [31]. This may be useful on some platforms when processing a partial message.

Internally, `overwrite` uses a `duplex` object. This fact was used in analysis by Bertoni and al. [31] who proved that the security of `overwrite` is equivalent to that of the `Sponge` construction with the same parameter, but at a cost of 2 digits of rate (or equivalently, of capacity): one for the padding rule (assuming $\texttt{pad10}^*$ is used) and one for the *frame digit* (this frame digit is equal to 1 for the last block and 0 for all other blocks).

Padding rules. Bertoni and al. [31] consider *sponge-compliant* padding rules that append a string that is fully determined by the length of M and the block length r. They only consider padding rules suitable for online mode of operation when the length of M is not known in advance.

For the padding rule we shall use the following notation: the padding of a message M to a sequence of r-digit blocks is denoted by $M \parallel \texttt{pad}[r](|M|)$.

A padding rule is sponge-compliant [30] if it never results in the empty string and if it satisfies following criterion:

$$\forall n \geqslant 0, \forall M, M' \in \mathbf{A}^* : M \neq M' \rightarrow M \parallel \texttt{pad}[r](|M|) \neq M' \parallel \texttt{pad}[r](|M'|) \parallel 0^{nr}.$$

As a sufficient condition, a padding rule that is reversible, non-empty and such that the last block must be non-zero, is sponge-compliant [31].

The most compact padding rule is the well-known *simple reversible padding*. This padding, denoted $\texttt{pad10}^*$, appends a single 1 and the smallest number of zeroes such that the length of the result is a multiple of the required block length. The simple reversible padding has only one digit of overhead.

When the length of M is known in advance it can be prepended before the message M: $\texttt{encode}(|M|) \parallel M$ where $\texttt{encode} : \mathbb{N} \rightarrow \mathbf{A}^*$ is a prefix code. A padding rule of this special form satisfies formal conditions on sponge-compliant padding:

$$\forall n \geqslant 0, \forall M, M' \in \mathbf{A}^* : M \neq M' \rightarrow \texttt{encode}(|M|) \parallel M \neq \texttt{encode}(|M'|) \parallel M' \parallel 0^{nr}.$$

When considering the security of a set of sponge constructions that make use of the same permutation F but with different rates, simple reversible padding is not sufficient [31]. As a solution, Bertoni and al. [31] propose the *multi-rate padding*. This padding, denoted $\texttt{pad10}^*1$, appends a single 1, the smallest number $t \geqslant 0$ of zeroes and a single 1 such that the length of the result is a multiple of the required block length. The multi-rate padding has two digit of overhead.

Domain separation. Domain separaion is used to distinguish between different uses of the same construction. It is usually done by prefixing input string with some control data. For example, adding one digit as a prefix allows to implement two oracles $\mathcal{O}_0$ and $\mathcal{O}_1$ one the basis of the same oracle $\mathcal{O}$.

In [30], an universal domain separation mechanism is proposed:

$$\mathcal{O}_{NS}[ns](data) := \mathcal{O}(\texttt{UTF8}(ns) \parallel 0^8 \parallel \texttt{encode}_{ns}(data)).$$

Here ns is a namespace defining a domain, $\texttt{UTF8}(ns)$ encodes a string ns as a byte string, $\texttt{encode}_{ns}$ is defined by an owner of the namespace ns.

With respect to AE the domains could separate key stream blocks and tag blocks.

Frame digits. Frame digits encode additional information appended to input blocks and usually serve the following purposes:

- as part of padding to indicate whether an input block is complete or incomplete;
- to distinguish between different domains in which output blocks will be used: as part of gamma stream or as authenticating tag;
- for input blocks decodability to distinguish between different types of input blocks: plaintext or additional data;
- to encode stage of operation of an AE scheme.

2.2.6 Generic security

The security of the `Sponge` construction is based on the assumption that the underlying function F is pseudorandom. Bertoni et al. [30] considered security of `Sponge` construction under generic attacks which do not make any weakness assumptions regarding the underlying function F. It is assumed that an adversary can make up to N calls to F (and to F^{-1} if F is a permutation). The estimates presented below are for binary alphabet.

The following generic attacks were considered:

1. *Inner collision*: find two strings such that their absorption results in the same inner state (outer states may differ). Success probability is $N^2/2^{c+1}$.
2. *Path finding*: find a string such that it's absorption results in a given inner state. Success probability is $N^2/2^{c+2}$ when F is permutation.
3. *Detect cycle*: detect cycles in outputs corresponding to valid input strings. Success probability is $N^2/2^{c+r+1}$ when F is transformation.
4. *State recovery*: for a given string Z such that $Z = S.Squeeze(S, |Z|)$ find state S. Success probability is $N/2^c$.
5. *Output binding*: for a given string Z find state S such that $S.Squeeze(S, |Z|) = Z$. Success probability is $N/2^{|Z|-r}$.

The vulnerability of the `Sponge` construction with respect to these attacks is due to its finite state and hence they do not apply to a random oracle. The success probabilities of generic attacks impose upper limits to the resistance the `Sponge` construction can offer.

The minimal expected number N of calls to F for one attack to succeed is $O(2^{c/2})$.

When sponge is used as hash function problems $\texttt{Inv}[h]$, $\texttt{2Pre}[h]$ and $\texttt{Coll}[h]$ introduced in 1.3.2 must be considered. Generic attacks can be applied to solve these problems.

1. `Inv`$[h]$: A pre-image can be obtained by binding an output string to a state and subsequently finding a path to that state. The expected workload for finding a pre-image is $2^n + 2^{c-1}$ where n is hash value length.

2. `2Pre`$[h]$: A second pre-image for a given input message can be obtained if a second path is be found to the inner state after absorption all but last blocks of the padded message. If F is a random permutation the expected workload is between $2^{(c+4)/2}$ and $2^{(c+3)/2}$.

3. `Coll`$[h]$: Finding inner collision results in collision of output strings. In a random sponge, the expected workload to generate an inner collision is of the order $2^{(c+3)/2}$. In a random oracle truncated to n digits, the expected workload to generate an output collision is of the order $2^{(n+3)/2}$.

This suggests that the security of unkeyed sponge constructions is determined by capacity c as $2^{c/2}$ operations.

For keyed modes the following attacks are considered.

1. *Predict keystream*: For an observed part of a keystream the rest of the keystream can be predicted after successful state recovery. The expected workload of this attack is 2^c.

2. *Forge tag*: For a set of observed message-tag pairs tag for a new message can be forged after successful state recovery. The expected workload is $2^{r+c-n}/(m-1)$ where $m+1$ is the number of chosen messages and n is the length of tag.

Bertoni et al. [30] established upper bounds on the advantage of an adversary distinguishing random oracle from the `Sponge` construction with a random function F in settings where the adversary does or does not have access to F. These bounds agree with the security bounds obtained from the generic attacks. It was also shown equivalence of the `Duplex` and `Sponge` constructions.

Jovanovic et. al [123] proved that the generic security level of keyed sponge constructions is higher than unkeyed and is lower bounded by

$$\min(2^{(r+c)/2}, 2^c, 2^{|K|}).$$

Although a fixed function F with a publicly available specification is not random, the assumption can be refined as follows: F does not have inner collisions [30].

2.2.7 The post-quantum security

Unruh et al. [67] investigated the post-quantum security of hash functions based on the classic sponge construction with outer state absorption (see 2.2). It was shown that in many applications post-quantum collision resistance is not enough and a notion of collapsing hash function was introduced as a strengthening of collision resistance. In [67] it was showed that the classic sponge construction with outer state absorption is collapsing under suitable assumptions about the underlying function F. In particular, if F is a random function, the classic sponge construction is collapsing and in consequence quantum collision-resistant. For the sponge construction, the complexity of the quantum algorithm for finding collisions asymptotically matches the complexity implied by collision resistance [68] .

In [142] was provided a security analysis of the full-state keyed sponge (FKS) and full-state keyed duplex (FKD) construction (see 2.2.4) in the quantum model. In this model, by utilizing the Simon's algorithm, proposed an universal forgery attack and key recovery attack to FKS and FKD with complexity of $O(c)$. Such attacks show that FKS and FKD are weak in the quantum model.

In Table 2.1 we summarize post-quantum strength of sponge-based constructions proposed for use in MAM2. These constructions are hash functions and AE algorithms. Hash functions are used at different stages of MSS signatures (see 3.4). For each sponge-based construction a corresponding set of related problems is considered. The parameters which impose security bounds are the following: k — length of key in trits, n — length of hash-value or authentication tag in trits, c — sponge capacity in trits.

Note that pre-quantum algorithms which solve the terget problems have higher complexity and thus are less efficient than post-quantum analogues for the same parameters k, n, c. Note also that for MAM2 we propose one security level for all cryptographic constructions.

Table 2.1: Post-Quantum strength of sponge-based problems

Construction	k	n	c	Problem	Strength
H (message hashing, see 1.3.2, 3.5)	–	234	237	$\texttt{Coll}[H]$	$\min(3^{n/3}, 3^{c/3}) = 3^{78}$
h (WOTS chains, see 1.3.2, 3.5)	–	162	237	$\texttt{Inv}[h, w-1]$, $\texttt{2Pre}[h, w-1]$, $\texttt{Hook}[h]$	$\min(3^{n/2}, 3^{c/3}) = 3^{79}$
h^* (WOTS public key shortening, see 1.3.2, 3.5)	–	243	237	$\texttt{Coll}[h^*]$	$\min(3^{n/3}, 3^{c/3}) = 3^{79}$
h_T (MSS authentication path, see 3.2, 3.4)	–	243	237	$\texttt{Coll}[h_T]$	$\min(3^{n/3}, 3^{c/3}) = 3^{79}$
AE (encryption and MAC, this chapter)	243	243	237	Key recovery, tag forgery	$\min(3^{k/2}, 3^{n}, 3^{c/3}) = 3^{79}$

2.3 Sponge-based authenticated encryption

Usually the inputs for the sponge-based authenticated encryption algorithm $\overline{\mathcal{E}}$ are a secret key K, an additional associated data A and a plaintext message M, where any of A, M can be empty (that is, of length 0). The output of the algorithm $\overline{\mathcal{E}}$ is an authenticated ciphertext C of exactly the same length as the plaintext message M, and an authentication tag T, which authenticates both A and M:

$$\overline{\mathcal{E}}(K, A, M) = (C, T).$$

The decryption and verification algorithm $\overline{\mathcal{D}}$ takes as input a secret key K, associated data A, a ciphertext C, and a tag T and outputs the plaintext message M if the verification of the tag is correct or the symbol $\perp$ if the verification of the tag fails:

$$\overline{\mathcal{D}}(K, A, C, T) \in \{M, \perp\}.$$

Internally, algorithms $\overline{\mathcal{E}}$ and $\overline{\mathcal{D}}$ use the `Duplex` construction or it's variant and maintain the state between calls in secret (or at least it's inner part).

The algorithms $\overline{\mathcal{E}}$ and $\overline{\mathcal{D}}$ can be parametrized by the key length, a tag length and other parameters.

AE can be defined such that the sender of a long plaintext can emit parts of the ciphertext before having read the entire plaintext (i.e., *on-line* scheme), or such that the sender always must read the entire plaintext before it can start sending out the first part of the ciphertext (i.e., *off-line* scheme) [1]. Note that the above definition for on-line and off-line AE considers only the encryption operation. This is due to the fact that the decryption operation must verify the tag of the all message and can return an error if the tag is incorrect. If this was not the case then an attacker could simply ask for the decryption of a ciphertext of his choice, while associating any arbitrary tag with the ciphertext [5].

In AE, a tag can be *intermediate* [31]. The intermediate tags allow the receiver to detect early if parts of a decrypted message are invalid, which saves computations when authenticating large messages [1]. Intermediate tags support some form of on-line decryption. If a long ciphertext $C = ((C_0, T_0), (C_1, T_1), \ldots)$ can be split into authenticated fragments (C_i, T_i), one can securely release the partial decryption of $(C_0, T_0), \ldots, (C_{i-1}, T_{i-1})$ before having read (C_i, T_i).

Usually, the sponge-based AE is *inverse-free*, i.e. it does not require either its underlying primitive's inverse operation (e.g. the inverse permutation for permutation-based schemes). It can save memory and area resources.

In [30], the following security requirements are imposed on sponge-based AE under the assumption that a key K is chosen secretly and uniformly over $|K|$ digits:

1. *Key recovery infeasibility.* The success probability of finding the key in an attack with effort equivalent to trying n key values is not above $n|\mathbf{A}|^{-|K|}$.

2. *Tag forgery infeasibility.* In the absence of key recovery, the success probability of tag forgery for any chosen (A, M) is $|\mathbf{A}|^{-|T|}$, even for an adversary that is given the corresponding cryptogram C and is given the outputs (C_i, T_i) corresponding to any set of adaptively chosen inputs (A_i, M_i) with the only restriction that $(A_i, M_i) \neq (A, M)$.

3. *Plaintext recovery infeasibility.* The most efficient method to gain information about M (excluding its length), given an output (C, T) corresponding to input (A, M) with chosen A but unknown M, is key recovery, even for an adversary that is given the outputs (C_i, T_i) corresponding to adaptively chosen inputs (A_i, M_i) with $A_i \neq A$.

Plaintext recovery infeasibility relies on the fact that for a given K there are no two inputs with equal data header A and different plaintext message M [30]. The uniqueness of the header A is as critical for security. Note that tag forgery does not rely on this.

2.4 Basic modes for authenticated encryption

2.4.1 The `SpongeWrap` mode

The `SpongeWrap` mode [31] is based on the `Duplex` construction.

The inputs for SpongeWrap are a secret key K, a header A (also referred as additional authenticated data), a plaintext M (for wrapping) or a ciphertext C (for unwrapping). Additionally for unwrapping a tag T is used as an input.

Before being forwarded to the duplex object, every key, header block, plaintext/ciphertext block is extended with a frame digit, providing domain separation resulting in protection against generic attacks.

Note that extended inputs are padded before processing. If header or plaintext/ciphertext are the empty string, they are treated as having a single block consisting of the empty string [30].

When processing inputs, firstly, key K is loaded into the state. Next, padded header A is absorbed into the state. After that the plaintext/ciphertext divided into blocks and then the encryption/decryption runs. Upon completion of processing of all plaintext/ciphertext blocks, the duplex object is call (with zero input data) until all the l-digit tag is squeezed from internal the state. In decryption mode the squeezed tag is compared with the received tag in order to check if the received tag is valid [185]. The mode is illustrated in Figure 2.6.

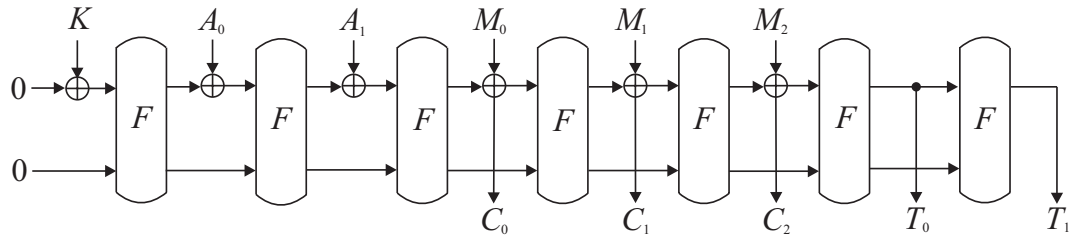

Figure 2.6: The SpongeWrap mode

SpongeWrap supports sessions, allowing the processing of several messages (each with associated data), where the tag for each message authenticates the full sequence of messages rather than only the message to which it was appended [35].

The security of SpongeWrap is based on the security of the sponge construction [31]. The mode can guarantee confidentiality if for the same key and different messages the header is unique.

2.4.2 The DuplexWrap mode

The DuplexWrap mode is very similar to SpongeWrap. The main differences from SpongeWrap are the following [35]:

- DuplexWrap puts no restriction in the way the input strings are cut into blocks;
- in the case only authentication is needed, the body can be absent and DuplexWrap avoids one call to the duplex object compared to SpongeWrap;
- for uniformity of the description, DuplexWrap does not propose a specific initialization call for absorbing the key (in the first wrap/unwrap call, the header must contain the secret key but can also contain a nonce and/or associated data);

  DuplexWrap provides an explicit "forget" call that processes the state in an irreversible way so as to ensure forward secrecy;

– `DuplexWrap` uses two frame digits per duplex call instead of one in `SpongeWrap`;

– `DuplexWrap` is instantiated explicitly with the multi-rate padding.

The security of the `DuplexWrap` are proved in [35].

2.4.3 The `MonkeyWrap` mode

The `MonkeyWrap` mode [37] is built on the `MonkeyDuplex` construction. The main difference from `MonkeyDuplex` is the following [37]: `MonkeyWrap` makes different rounds of the permutation F when transitioning to tag generation than in other cases. This mode also has a simpler way to load the key and a nonce during the initialization.

For the `MonkeyWrap` mode the wrapping is considered as a process that takes as input a header A and a plaintext message M and that returns a ciphertext C and a tag T. In turn, the unwrapping is considered as a process that takes as input a header A a ciphertext C and a tag T and returning the plaintext message M if the tag T is correct. The wrapping and unwrapping is initialized by loading a key K and a nonce N. After initialization, two frame digits are appended to each input block and the multi-rate padding are used.

The following refinements of the `MonkeyDuplex` construction are valid for `MonkeyWrap`. After the key and nonce have been put in the state the permutation F with n_{start} rounds are applied. In a duplex call n_{step} or n_{stride} rounds of the permutation F are used. Using n_{step} or n_{stride} rounds are serve different purposes. Both aim at providing resistance against state retrieval, but in addition, using n_{stride} rounds also aims at providing resistance against output forgery [37]. Hence this requires that $n_{step} < n_{stride}$. Typically we also have $n_{stride} < n_{start}$. The `MonkeyWrap` mode is illusrated in Figure 2.7, where $n_1 = n_{start}$, $n_2 = n_{step}$ and $n_3 = n_{stride}$.

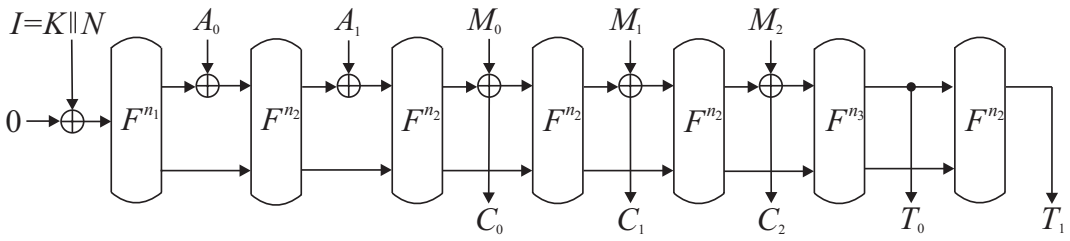

Figure 2.7: The mode `MonkeyWrap`

The generic security of `MonkeyWrap` is similar to one of `SpongeWrap` [37].

2.5 AE algorithms of MAM

2.5.1 General

AE algorithms are specifically crafted for IOTA MAM purposes: messages are encrypted and signed. IOTA MAM AE is based on a simplified duplex construction over Curl's permutation $F\colon \mathbf{T}^{729} \to \mathbf{T}^{729}$ with overwrite mode and is similar to the `SpongeWrap` mode. AE uses an internal state of width $b = 729$ trits with rate $r = 243$ and capacity $c = 486$. AE uses outer-state key initialization. While the implementation allows to use keys of any size, usually

keys of size 243 trits are used. IOTA MAM AE wrapping algorithm is hardwired with Merkle tree signature. Public key (the Merkle tree root together with the index of the current private key) is used as a unique initialization vector and additional authenticated data. Depending on the signature security level tag size varies between 27, 54, and 81 trytes. As sizes of the plaintext and public key are used as additional authenticated data trivial padding and truncation algorithms are used.

2.5.2 Reconstruction

Here we present the reconstruction of MAM algorithms. The main algorithms which "mask" and "unmask" messages are denoted here as `SE.Wrap` and `SE.Unwrap`.

Auxiliary algorithms. Our reconstruction makes use of the following auxiliary algorithms which we only briefly describe here.

`Init`() initializes the state S with zero trits.

`Absorb`(X) overwrites the first $|X| \leqslant 243$ trits of the outer state with message X and applies transformation F to the updated state.

`Rate`(n) returns the first $n \leqslant 243$ trits of the outer state, the default value for n is 243 and can be omitted.

`Num2Tri`(n) represents an integer number n as a string of trits.[1] A variable-length redundant coding is employed. The length of the code-word is roughly $\lceil \log_3(2n+1) \rceil_3 + \lceil \log_3 2^{\lceil \log_3(2n+1) \rceil / 3} \rceil_3$ where $\lceil \cdot \rceil_3$ denotes rounding up to a nearest multiple of 3.

`Tri2Num`(t) parses prefix of a string of trits t as an integer number n and returns pair (n, t') where t' is the rest of the string t. The encoding of a number is not unique: there exist code-words that are decoded to the same number n. The size of a code-word and possible overflows are not checked in the reference implementation.

`Take`(X) splits a string X into two parts X' and X'', such that $X = X' \parallel X''$ and $|X'| = min\{243, |X|\}$.

`Next`(N) takes a nonce $N \in \mathbf{T}^{81}$ and returns a new unique nonce N'. `Next` from the reference implementation increments nonce N in a specific way.

`Check`(T) takes a 243-trit string T, converts it into a string of trytes and calculates their sum as integers. If the sum of the first 27 trytes equals zero then 1 is returned. If the sum of the first 54 trytes equals zero then 2 is returned. If the sum of all the 81 trytes equals zero then 3 is returned. Otherwise 0 is returned.

`MAM.Sign`(sk, H) signs a hash value $H \in \mathbf{T}^{L \times 81}$ with a private key $sk \in \mathbf{T}^{L \times 27 \times 243}$ (see 3.3.2.2). The acceptable length of H is determined by a security level L of sk. The signature consists of WOTS signature $sig \in \mathbf{T}^{L \times 27 \times 243}$, the height of Merkle tree $N \in \mathbb{N}$ and authentication path $P \in \mathbf{T}^{N \times 243}$ of the WOTS public key in Merkle tree.

`MAM.Verify`($root, sig, H$) verifies a signature sig of a hash value H with a public key $root$ (see 3.3.2.3). The security level is determined by the value of H and must match the security level of $root$. sig consists of of $WOTS$ signature and augmenting path in Merkle tree. $root$ is Merkle tree root.

Message part encryption algorithm. Algorithm `Encr` performs encryption of a message part.

[1] The reference implementation uses platform-dependent type `isize` for n which can hold signed 32 or 64 bit interger.

Algorithm Encr (encrypt message part)

Input:
$X \in \mathbf{T}^*$ — plaintext.
Output:
$Y \in \mathbf{T}^*$ — ciphertext.
State:
$S \in \mathbf{T}^{729}$ — internal state.

Steps:

1. $Y \leftarrow \epsilon$.
2. $i \leftarrow 0$.
3. While $0 < |X|$ do:
   (a) $i \leftarrow i + 1$.
   (b) $(X_i, X) \leftarrow \texttt{Take}(X)$.
   (c) $Z \leftarrow \texttt{Rate}(|X_i|)$.
   (d) $Y_i \leftarrow X_i \oplus Z$.
   (e) $Y \leftarrow Y \parallel Y_i$.
   (f) $\texttt{Absorb}(X_i)$.
4. Return (Y, S).

The secret state S is updated in the process.

Message part decryption algorithm. Algorithm `Decr` performs decryption of a message part.

Algorithm Decr (decrypt message part)

Input:
$Y \in \mathbf{T}^*$ — ciphertext.
Output:
$X \in \mathbf{T}^*$ — plaintext.
State:
$S \in \mathbf{T}^{729}$ — internal state.

Steps:

1. $X \leftarrow \epsilon$.
2. $i \leftarrow 0$.
3. While $0 < |Y|$ do:

(a) $i \leftarrow i + 1$.

(b) $(Y_i, Y) \leftarrow \texttt{Take}(Y)$.

(c) $Z \leftarrow \texttt{Rate}(|Y_i|)$.

(d) $X_i \leftarrow Y_i \ominus Z$.

(e) $X \leftarrow X \parallel X_i$.

(f) $\texttt{Absorb}(X_i)$.

4. Return (X, S).

The secret state S is updated in the process.

Search for nonce. Algorithm `Search` is a bruteforce algorithm for finding nonce.

Algorithm Search (find nonce)

Input:
$S \in \mathbf{T}^{729}$ — internal state;
$L \in \{1, 2, 3\}$ — security level.
Output: $nonce \in \mathbf{T}^{81}$ — nonce.
Steps:

1. $N \leftarrow \texttt{Rate}()$

2. Do:

   (a) $N \leftarrow \texttt{Next}(N)$.

   (b) $S' \leftarrow S$.

   (c) $\texttt{Absorb}(N)$.

   (d) $T \leftarrow \texttt{Rate}()$.

   (e) $S \leftarrow S'$.

3. While $\texttt{Check}(T) \neq L$.

4. Return N.

At the step 2a an arbitrary nonce is picked with `Next`. `Next` from the reference implementation modifies nonce as follows: the last 27 trits of nonce N are incremented. But nonce can be picked in any (random) way.

`Check` at the step 3 splits T into trytes and calculates their sum as integers. If the sum is zero then `Check` returns 1 else 0 is returned.

MAM message wrapping algorithm. The algorithm `SE.Wrap` produces a MAM message containing an encrypted and signed payload.

Algorithm SE.Wrap (create MAM message)

Input:
$PSK \in \mathbf{T}^{243}$ — secret pre-shared key;
$X \in \mathbf{T}^{*}$ — payload;

$L \in \{1, 2, 3\}$ — security level;
N — the Merkle tree height;
$sk \in \mathbf{T}^{L \times 27 \times 243}$ — sender's current private WOTS key;
$skn \in \mathbb{N}$ — zero-based index of the private key within the Merkle tree;
$root \in \mathbf{T}^{243}$ — sender's public key — root of the Merkle tree;
$P \in \mathbf{T}^{243 \times N}$ — authentication path containing proof for sender's current public WOTS key;
$root'$ — the next sender's public key — root of the new Merkle tree.
Output: $Y \in \mathbf{T}^*$ — MAM message.
State: $S \in \mathbf{T}^{729}$ — state of the sponge function is reused and destroyed on exit.
Steps:

1. `Init`().
2. `Absorb`(PSK).
3. `Absorb`($root$).
4. `Absorb`(`Num2Tri`(skn) $\|$ `Num2Tri`($|X|$)).
5. $Y \leftarrow$ `Num2Tri`(skn) $\|$ `Num2Tri`($|X|$).
6. $Z \leftarrow$ `Encr`($root' \| X$), $Y \leftarrow Y \| Z$.
7. $nonce \leftarrow$ `Search`(S, L).
8. $Z \leftarrow$ `Encr`($nonce$), $Y \leftarrow Y \| Z$.
9. $T \leftarrow$ `Rate`($L * 81$).
10. $sig \leftarrow$ `MAM.Sign`(sk, T).
11. $Z \leftarrow$ `Encr`(sig), $Y \leftarrow Y \| Z$.
12. $Z \leftarrow$ `Encr`(`Num2Tri`(N) $\| P$), $Y \leftarrow Y \| Z$.
13. Return Y.

Using `Next` from the reference implementation in `Search` at step 7 allows to represent the encrypted nonce Z at step 8 as $Z = nonce \oplus nonce \oplus \epsilon$, where ϵ has a small Hamming weight. Hence $nonce$ can be restored from Z with a small error. This suggests that the encryption of $nonce$ is redundant.

MAM message unwrapping algorithm. The algorithm `SE.Unwrap` decrypts payload of a MAM message and verifies it's signature.

Algorithm SE.Unwrap (parse MAM message)

Input:
$PSK \in \mathbf{T}^{243}$ — secret preshared key;
$Y \in \mathbf{T}^*$ — MAM message;
$root$ — sender's public key – root of the Merkle tree.

Output:
$X \in \mathbf{T}^*$ — payload;
$root'$ — next sender's public key – root of the new Merkle tree – new channel ID.
State: $S \in \mathbf{T}^{729}$ — state of the sponge function is reused and destroyed on exit.
Steps:

1. $\texttt{Init}()$.
2. $\texttt{Absorb}(PSK)$.
3. $\texttt{Absorb}(root)$.
4. $(skn, Z) \leftarrow \texttt{Tri2Num}(Y)$; $Y \leftarrow Z$.
5. $(|X|, Z) \leftarrow \texttt{Tri2Num}(Y)$; $Y \leftarrow Z$.
6. $\texttt{Absorb}(\texttt{Num2Tri}(skn) \parallel \texttt{Num2Tri}(|X|))$.
7. Parse $Y' \parallel Z \leftarrow Y$ such that $|Y'| = |root| + |X|$.
8. $(root' \parallel X) \leftarrow \texttt{Decr}(Y')$, $Y \leftarrow Y \parallel Z$.
9. Parse $Y' \parallel Z \leftarrow Y$ such that $|Y'| = 243$.
10. $nonce \leftarrow \texttt{Decr}(Y')$, $Y \leftarrow Y \parallel Z$.
11. $T \leftarrow \texttt{Rate}()$.
12. $L \leftarrow \texttt{Check}(T)$.
13. $T' \leftarrow T[\ldots L * 81)$.
14. $sig \leftarrow \texttt{Decr}(Y)$.
15. If $\texttt{MAM.Verify}(root, sig, T') = 0$ then return $\perp$.
16. Return $(root', X)$.

After the body is encrypted a certain random nonce is added and encrypted. The rate of the current state of the duplex construction is interpreted as authentication tag. The tag is signed with Merkle-tree WOTS signature and the signature value is then encrypted. As the tag is not hashed and is interpreted as a hash-value to-be-signed, hence it must have sum of it's trytes equal to zero. In order to get such tag a nonce is introduced at the last step of encryption prior to getting the tag. Although the nonce is calculated deterministically it is not checked when MAM message is decrypted. This may potentially give an attacker additional advantage.

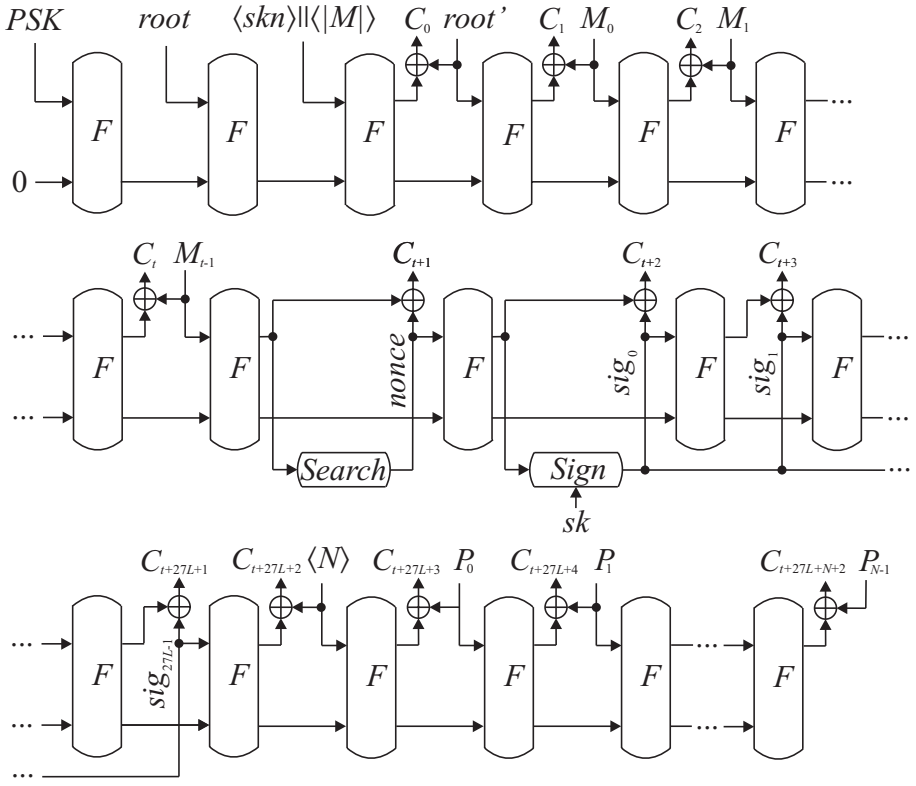

Figure 2.8: The IOTA MAM AE

2.5.3 Comparison of IOTA MAM AE and sponge-based AE

IOTA MAM AE is based on the duplex construction. The IOTA MAM AE is illustrated in Figure 2.8. In the figure we use the notation of Section 2.5.2.

In MAM encryption key consists of two parts PSK and $root$, their usage depends on the channel mode (see 1.5). PSK can either have a constant value or can be a pre-shared key. $root$ is a Merkle tree root of MSS and can either be secret or public. Each key part is loaded using `Absorb` i.e. it takes two transformations F to load key.

Next, MAM message header is processed. A header contains private key number skn and message payload length $|X|$. Header is not encrypted. In MAM for each message the $(root, skn)$ pair is unique.

Next, message body is encrypted. Message body consists of the sender's next public key $root'$ and payload X.

IOTA's documentation on MAM does not explicitly state what can serve as a payload X. One of the possible use cases for MAM is publishing new IOTA addresses.

Next, the $nonce$ value is created using the `Search` algorithm and encrypted. $nonce$ is required for the purpose of signature algorithm. The `Search` algorithm returns a pseudorandom nonce value such that corresponding authentication tag has sum of it's trytes equal to zero.

Finally, authentication tag is signed with skn-th private key. Signature security level is an input of message wrapping algorithm. Message unwrapping algorithm determines security level implicitly using procedure `Check` and authentication tag. Signature including authentication path in Merkle tree is added and encrypted after WOTS part of the signature.

The mode of MAM AE is similar to `SpongeWrap`, but has the following distinctive features:

1. MAM as a part of IOTA is ternary logic oriented meaning that MAM AE algorithms transform ternary strings.

2. MAM fuses sponge-based AE and MSS signature to protect messages. Usually, AE provides integrity but not authenticity and signature algorithm is applied separately from AE algorithm. Also, AE can effectively be disabled by using the publicly fixed key.

3. Initially, when a channel is created, skn equals 0. With each next message skn is incremented and the message is signed with a corresponding private key. Inclusion of the next public key $root'$ into a message allows recipients to safely switch to a new channel.

4. Since IOTA MAM AE is based on the `SpongeWrap`, it inherits many of its properties in misuse settings. For example, repeating the header once for two different messages would not only reveal the difference between the first differing plaintext blocks but also compromise WOTS private key.

5. Two calls to F are required in order to load key which is inefficient.

6. MAM messages have a strict format with a certain order of message parts. As a result domain separation mechanisms are not relevant.

7. Absorbing is done by overwriting the outer state by the input rather than XORing it in.

8. MAM AE does not use padding while processing message parts. Instead the length of the current variable-length part is prepended and processed. MAM message has two such parts: payload and authentication path. The length of the payload is specified in the header. The length of authentication path is a part of signature which is encrypted.

9. MAM AE doesn't use `forgetting` operation (e.g. integrated forgetting operation from Keyak v.2.2 [36], the RATCHET operation from Strobe [107]). The operation is designed to prevent rollback attacks (see 2.6.1). If the last state of some message becomes known (via an exploit, for example) then the secret key could be recovered and all the messages of the channel become compromised (could be decrypted).

2.6 Analysis of MAM AE

2.6.1 The rollback attack

The rollback attack proceeds as follows. Suppose an attacker recovers the internal state of AE during data processing, perhaps through an application exploit. She can reverse AE steps to learn earlier states and recover the key. In case of key recovery, the attacker can decrypt all messages that have been encrypted on this key. We estimate the applicability of the rollback attack to the algorithm `SE.Wrap`, used to protect the channel in the "private" mode.

Let's denote the header by $hdr = \texttt{NumToTri}(skn) \parallel \texttt{NumToTri}(|M|)$, and the length of hdr in trits by l. Let $|PSK| = |root| = |root'| = r$. Then the steps of the `SE.Wrap` algorithm wrapping payload $M = M_0 \parallel M_1 \parallel \ldots \parallel M_{t-1}$, $|M_{t-1}| = r$, and producing ciphertext

$C = C_0 \parallel C_1 \parallel \ldots \parallel C_{t-1}$ can be written as follows (see 2.5.2):

$$
\begin{aligned}
S_0 &= F(PSK \parallel 0^c), \\
S_1 &= F(root \parallel S_0[r \ldots]), \\
S_2 = F(hdr \parallel S_1[l \ldots]), &\quad C_0 = S_2[\ldots r] \oplus root', \\
S_3 = f(root' \parallel S_2[r \ldots]), &\quad C_0 = S_3[\ldots r] \oplus M_0, \\
S_4 = F(M_0 \parallel S_3[r \ldots]), &\quad C_1 = S_4[\ldots r] \oplus M_1, \\
&\ldots \\
S_{t+2} = F(M_{t-2} \parallel S_{t+1}[r \ldots]), &\quad C_{t-1} = S_{t+2}[\ldots r] \oplus M_{t-1}.
\end{aligned}
$$

Suppose an attacker knows hdr, $root'$, M, C and S_{t+2}. As F is an invertible permutation the attacker can recover all previous states up to S_2. In its turn, knowing state S_2 the attacker can recover state $S'_1 = f^{-1}(S_2)$, such that $S'_1[l \ldots] = S_1[l \ldots]$. Recovering key $root = F^{-1}(S_1)[..r]$ requires the attacker to bruteforce all 3^l possible values for $S_1[\ldots l]$. Difficulty of finding such $S_1[\ldots l]$ depends on the header hdr length which in turn depends on the `NumToTri` code word length. `NumToTri` output lengths for small inputs are presented in table 2.2. In particular, if $skn \leqslant 364$ and $|M| \leqslant 9841$, then $S_1[\ldots l]$ can have at most $3^{8+12} = 3^{20} \approx 2^{31.7}$ values.

Table 2.2: The values of `NumToTri`

n	$\lvert$`NumToTri`$(n)\rvert$
0 – 13	4
14 – 364	8
365 – 9841	12
9842 – 265720	16

Thus, for known hdr, $roor'$, M and C a recovery of the internal state during processing M does directly lead to a recovery of the key $root$ with the insignificant computational complexity.

In order to counteract this attack it is sufficient that header hdr is padded up to a full block of length r before being processed by algorithm `SE.Wrap`.

2.6.2 The time-memory tradeoff attack

In case when MAM AE is used to encrypt a known plaintext algorithm `SE.Wrap` can be viewed as a stream cipher producing keystream in blocks of size $r = 243$ trits. In this setting, the inner state S_c or the key K is unknown.

One of the most widely-spream universal attacks on stream ciphers are the time-memory tradeoff (TMTO) attacks. The first TMTO attack was introduced by M. Hellman in his paper [109]. Hellman's ideas, originally proposed for attacking DES, were extended to various cryptosystems.

In general these attacks are used to invert a one-way function, i.e. to identify an unknown preimage of a known image. Such one-way functions map internal state or key and initialization vector of the algorithm into it's output stream called prefix [114]. The length of the prefix depends on the class of the attack. If the attack is used to recover an internal state then the

prefix length is equal to the length of the internal state. And if it is used to recover a key then the prefix length is the sum of the lengths of key and IV.

If an internal state was recovered as a result of an attack then in order to recover key the state must be rolled back. Rollback attack on MAM AE was considered in 2.6.1. Note, MAM AE state rollback complexity is significantly lower than complexity of state recovery and can be ignored when considering applicability of TMTO attacks.

Prefixes used in an attack can correspond to the same key or different keys. Prefixes corresponding to the same key can overlap which allows to reduce total size of observable keystreams required for the attack to be successful.

Note, in MAM AE setting prefix is a concatenation of sequential output blocks and the header serves as IV.

TMTO attacks consist of two phases [81]:

1) precomputation phase in which an attacker creates data tables using the target algorithm being analyzed.

2) real-time phase (search phase) in which an attacker observes output prefixes and recovers internal state or key of the target algorithm using precomputed data tables.

TMTO attacks can be characterized with the following parameters [39]:

- N — search space cardinality (internal state set or key set cardinality);
- P — precomputation phase complexity;
- M — size of memory required for precomputed tables;
- T — real-time phase complexity;
- D — toatal size of prefixes required for an attack;
- π — attack success probability.

In order for the attack to succeed the parameters must satisfy certain tradeoff curve depending on the class of the attack.

Let's consider a few TMTO attacks and their applicability to MAM AE.

In [17] and [101] an attack (let's call it BG-attack) based on the birthday paradox is considered. It's purpose is to recover an initial state of the algorithm. Tradeoff curve of the BG-attack is $TM = N$, where $T = D$, $P = M = N/D$ and N — state set cardinality. The solution $T = M = D = P = N^{1/2}$ to this curve has success probability $\pi = 0.63$.

In [39] a modification of Hellman's attack (let's call it BS-attack) is considered. It's purpose is to recover an initial state of the algorithm. Tradeoff curve of the BS-attack is $TM^2D^2 = N^2$, where $1 \leqslant D^2 \leqslant T$, $P = N/D$ and N — state set cardinality. The solution $T = M = N^{1/2}$, $D = N^{1/4}$, $P = N^{3/4}$ to the tradeoff curve has success probability $\pi = 0.63$.

BG-attack and BS-attack are used to recover initial internal state. In case when the length of key and IV is less than that of internal state more efficient modifications of these attacks can be considered. For example, in [114] modified BG-attack and BS-attack recovering key using prefixes obtained with the same key but different IVs are considered. In MAM AE setting these modified attacks are not feasible as only an insignificant amount of messages can be encrypted using the same key but different headers.

In [81] a modification of BS-attack (let's call it DK-attack) recovering key using prefixes obtained with different keys and a fixed known IV is considered. Tradeoff curve of the DK-attack is $TM^2D^2 = N^2$, where $1 \leqslant D^2 \leqslant T$, $P = N/D$ and N — key set cardinality. The following are some solutions to this curve:

– $(P, D, M, T) = (N^{2/3}, N^{1/3}, N^{1/3}, N^{2/3})$;

– $(P, D, M, T) = (N^{3/4}, N^{1/4}, N^{1/2}, N^{1/2})$.

Success probability for these solutions is $\pi = 0.63$.

Table 2.3 shows characteristics of different attacks in MAM AE setting. Note, MAM AE has (see 1.5): inner state $S_c \in \mathbf{T}^{486}$, key $K \in \mathbf{T}^{486}$ (in "restricted" mode) or $K \in \mathbf{T}^{243}$ (in "private" mode).

Table 2.3: TMTO for MAM AE

Attack	N	T	M	D	P
BG-attack [17, 101] on $S_c \in \mathbf{T}^{486}$	3^{486}	3^{243}	3^{243}	3^{243}	3^{243}
BS-attack [39] on $S_c \in \mathbf{T}^{486}$	3^{486}	3^{243}	3^{243}	$3^{121.5}$	$3^{364.5}$
DK-attack [81] on $K \in \mathbf{T}^{486}$	3^{486}	3^{324}	3^{162}	3^{162}	3^{324}
DK-attack [81] on $K \in \mathbf{T}^{486}$	3^{486}	3^{243}	3^{243}	$3^{121.5}$	$3^{364.5}$
DK-attack [81] on $K \in \mathbf{T}^{243}$	3^{243}	3^{162}	3^{81}	3^{81}	3^{162}
DK-attack [81] on $K \in \mathbf{T}^{243}$	3^{243}	$3^{121.5}$	$3^{121.5}$	$3^{60.75}$	$3^{182.25}$

Table 2.3 shows that among TMTO attacks best characteristics has DK-attack on key $K \in \mathbf{T}^{243}$. The sizes of memory and data are $M = 3^{121.5} \approx 2^{192.57}$ and $D = 3^{60.75} \approx 2^{96.29}$, complexities of the real-time and precomputation phases are $T = 3^{121.5} \approx 2^{128.38}$ and $P = 3^{182.25} \approx 2^{288.86}$. These values are impractical which proves security of MAM AE against DK-attak.

2.6.3 The forgery attack

Let's consider applicability of the Forgery attack on the `SE.Wrap` algorithm. Cryptographical security of transformation F is assumed. Here the public mode (see 1.5) of operation of `SE.Wrap` algorithm is considered.

Let an attacker choose messages to be protected by `SE.Wrap`. In the public mode message payload is protected with digital signature using sender's private key. The attacker has no access to sender's private key. In the Forgery attack on MAM AE the attacker observes a protected message and needs to find a new message with a valid signature equal to that of the observed message. The forged message length may differ from the length of the original message. These conditions are consistent with the "Existential forgery" attack.

The notation introduced for description of `SE.Wrap` algorithm is used here. In particular, M denotes message payload, skn — MSS index of private key used to sign M, and `NumToTri`(n) — function mapping a natural number n into a sequence of trits. For simplicity let's assume that a single private key with $skn = 0$ is used to sign messages.

During our reasoning we will take into account the following MAM AE features (see 2.5.3):

– absorbing is done by overwriting the outer state by the input;

– the mode uses the length-padding scheme;

The forgery attack is applicable with a certain probability only if there exist values $l, l' \in \mathbb{N}$ and a string $\alpha \in \mathbf{T}^*$ such that:

$$l' > l, \quad \lfloor l/r \rfloor = \lfloor l'/r \rfloor, \quad l \mod r \neq 0, \tag{2.1}$$

$$0 < |\alpha| \leqslant r - |\texttt{NumToTri}(0)|, \tag{2.2}$$

$$\texttt{NumToTri}(l') = \texttt{NumToTri}(l) \parallel \alpha. \tag{2.3}$$

Indeed, let there exist l, l' and α satisfying (2.1) – (2.3). And let there exist a message M of length l and a message $M' = M \parallel \beta$ of length l', where β is chosen randomly and uniformly from $\mathbf{T}^{l'-l}$. Then the messages M and M' protected and signed with the same private key will have the same signature with probability $p = 3^{-(|\alpha|+|\beta|)}$. It follows, for example, that the attacker observing p^{-1} different messages M of length l will be able to construct a new message M' of length l' such that it's signature will coincide with a signature of one of the observed messages with probability approaching 1.

As such, applicability of the forgery attack on `SE.Wrap` algorithm depends on properties of `NumToTri` function. Let's show that for `NumToTri` implemented in MAM AE values l, l', and α satisfying (2.1) – (2.3) do not exist. In other words, the forgery attack on MAM AE is not applicable. Indeed, the contradiction to the (2.3) directly follows from `NumToTri` which implements prefix encoding. In particular, for any $\alpha \in \mathbf{T}^*$ there exist at most one $l \in \mathbb{N}$ such that $\texttt{NumToTri}(l)$ is a prefix of α.

2.7 AE algorithms of MAM2

Here we consider some tweakable settings for MAM2 AE.

1. *Basic pseudorandom function F.* As mentioned in 1.3.3 we only make assumption that F is secure and has interface $\mathbf{T}^{729} \to \mathbf{T}^{729}$. Also we do not make assumptions regarding the internal number of iterations of F although tweaking the number of iterations at certain steps might speed up MAM2 AE.

2. *Security level.* In Post-Quantum Epoch complexity of solving any cryptographic problem should be roughly $O(3^{81})$ quantum operations, or simply said — 81 trits.

3. *Key size.* According to current estimates (2.1) key size of 162 trits guarantees security level of 81 trits. This is the lower bound for key size in Post-Quantum Epoch. MAM2 AE key size will be 243 trits which guarantees additional security.

4. *Rate/capacity tradeoff.* The 486/243 trits variant seems reasonable: the capacity is large enough to provide 81 trits of security against post-quantum inner state collision attacks. This is the lower bound for capacity in Post-Quantum Epoch. Inner state can additionally hold padding and frame trits reducing security by several trits.

5. *Padding.* 10^* is the most compact sponge-compliant padding scheme. As rate is fixed there is no need to employ multi-rate padding.

6. *Absorption mode.* Overwrite mode has been shown to be as secure as sponge construction. Overwrite compared to Plus mode requires less trits to be stored between calls to F. It also has "forgetting" semantic when additional authenticated data is processed. Hence overwrite mode is preferable.

7. *Data processing mode.* Data elements are divided into categories: `KEY`, `DATA`, `HASH`, `TAG` and other. Data elements shall be non-empty and can be added to and extracted from state in any order.

   Key must be added to the state before processing critical data but may be added after processing associated data.

8. *Fork/merge support.* The current state may be duplicated and used in two different following operations. It may be used for advanced key loading.

9. *Parallel encryption.* Parallel encryption is not a crucial for IOTA MAM2 feature and is not supported.

10. *Mode of operation.* The idea is to build for MAM2 not only AE but also hash and pseudorandom data generation algorithms on the basis of sponge function. Corresponding operations similar to `absorb`, `squeeze` and `duplexing` must be supported. These operations should perform in uniform manner for keyed and unkeyed modes.

11. *Frame trits and domain separation.* As data update mode is quite relaxed domain separation of input and output data between calls to F is required. For each data element 3 frame trits indicate whether current block is complete or incomplete, whether current block is last or intermediate and which operation is performed.

12. *MAC tag size.* 81 trits can be used as tag. Tag can be signed and not be explicitly present in the message. In this case the size of tag equals to the size of hash value to be signed — 243 trits.

13. *MAC count with the same key.* 3^{81} texts can be authenticated using the same key. A larger number of tags imposes threat of tag forgery.

2.8 Informal specification of SpongeMAM2

2.8.1 General

SpongeMAM2 is a framework for building the following series of cryptographic algorithms:

- authenticated encryption (wrapping and unwrapping);
- encryption and decryption;
- message authentication codes (MAC);
- hashing;
- pseudorandom number generation.

The construction of SpongeMAM2 is based on duplex construction like `SpongeWrap`, but uses a overwriting mode for the absorption.

The core permutation F operate on a sponge state $S \in \mathbf{T}^{729}$, with a rate of $r = 486$ trits, a capacity of $c = 237$ trits and 6 control trits.

2.8.2 Padding

SpongeMAM2 has a block size of r trits. Each input X is split into the blocks X_1, X_2,...,X_n such that $|X_1| = \ldots = |X_{n-1}| = r$, $0 \leqslant |X_n| \leqslant r$. The padding process appends a single 1 and the smallest number of 0s to each block such that the length of the padded block is $r + 1$ trit. The last trit of the padded block used as the frame trit.

2.8.3 Control trits

SpongeMAM2 uses 6 control (frame) trits $t_1t_2t_3t'_1t'_2t'_3 = S[r \ldots r + 6)$. Trits $t_1t_2t_3$ describe the previous operation, and trits $t'_1t'_2t'_3$ describe the current operation. To be more exact, the trits t_1, t'_1 describe a block of input data processed during the target operation (either the last padding trit or indication of input data absence), the trits t_2, t'_2 describe a type of block of input or output data, the trits t_3, t'_3 describe a type of the operation.

The trits t_1, t'_1 have the values: 0 — an incomplete input block, 1 — a complete input block, $\overline{1}$ — no input block.

The trits t_2, t'_2 have the values: 0 — a block of the initial state, 1 — an intermediate block of this data element, $\overline{1}$ — a last block of this data element.

The trits t_3, t'_3 have the values: 0 — processing public data or generating a hash value, 1 — processing a secret key or generating pseudorandom numbers, $\overline{1}$ — processing plaintext/ciphertext or generating a tag (MAC).

2.8.4 Operations

SpongeMAM2 implements the following basic operations:

- Initialization;
- Key Loading;
- Associated Data Processing;
- Plaintext Processing;
- Ciphertext Processing;
- Tag (MAC) Unloading;
- Hash Unloading;
- Pseudorandom Number Generation.

Using these basic operations cryptographic algorithms presented in 2.8 are constructed.

All the operations use the same state S. The state is changed during the operations.

Initialization. The operation has no input and no output arguments. It is used to initialize state S:

$$S \leftarrow 0^{729}.$$

This operation can additionally be used to cleanup state S after performing other operations.

Key Loading. The operation has key $K \in \mathbf{T}^*$, $0 < |K| \leqslant r$, as an input argument and has no output arguments.

Key loading directly following the initialization is performed in the following way:

$$S \leftarrow K \| 1 \| 0^{r-|K|} \parallel \overline{1}1 \parallel S[r+3 \dots).$$

Otherwise key loading is performed in the following way:

$$S \leftarrow \begin{cases} S[\dots r+3) \parallel 0\overline{1}1 \parallel S[r+6 \dots), & \text{if } 0 < |K| < r, \\ S[\dots r+3) \parallel 1\overline{1}1 \parallel S[r+6 \dots), & \text{if } |K| = r, \end{cases}$$
$$S \leftarrow F(S),$$
$$S \leftarrow K \| 1 \| 0^{r-|K|} \parallel \overline{1}1 \parallel S[r+3 \dots).$$

Associated Data Processing. The operation has associated data $A \in \mathbf{T}^*$ split into blocks $A_1, \dots, A_n$ as input arguments and has no output arguments.

Let's consider the general case when the operation does not directly follows the initialization.

Each intermediate block A_i is processed as follows:

$$S \leftarrow S[\dots r+3) \parallel 110 \parallel S[r+6 \dots), \tag{2.4}$$
$$S \leftarrow F(S), \tag{2.5}$$
$$S \leftarrow A_i \parallel 110 \parallel S[r+3 \dots). \tag{2.6}$$

The last block A_n is processed as follows:

$$S \leftarrow \begin{cases} S[\dots r+3) \parallel 0\overline{1}0 \parallel S[r+6 \dots), & \text{if } 0 \leqslant |A_n| < r, \\ S[\dots r+3) \parallel 1\overline{1}0 \parallel S[r+6 \dots), & \text{if } |A_n| = r, \end{cases} \tag{2.7}$$
$$S \leftarrow F(S), \tag{2.8}$$
$$S \leftarrow A_n \parallel 10^{r-|A_n|} \parallel \overline{1}0 \parallel S[r+3 \dots). \tag{2.9}$$

When the operation is called directly after the initialization and the associated data consist of more than one block then the (2.4) and (2.5) steps of the first block processing are not executed.

When the operation is called directly after the initialization and the associated data consist of one block then the (2.7) and (2.8) steps are not executed.

Plaintext Processing. The operation has plaintext $M \in \mathbf{T}^*$ split into blocks $M_1, \dots, M_n$ as input arguments and ciphertext $C \in \mathbf{T}^*$ split into blocks $C_1, \dots, C_n$ such that $|C_i| = |M_i|$ as output arguments.

Each intermediate block M_i is processed as follows:

$$\begin{gathered} S \leftarrow S[\dots r+3) \parallel 11\overline{1} \parallel S[r+6\dots), \\ S \leftarrow F(S), \\ C_i \leftarrow S[\dots |M_i|) \oplus M_i, \\ S \leftarrow M_i \parallel 11\overline{1} \parallel S[r+3\dots). \end{gathered}$$

The last block M_t is processed as follows:

$$\begin{gathered} S \leftarrow \begin{cases} S[\dots r+3) \parallel 0\overline{11} \parallel S[r+6\dots), & \text{if } 0 \leqslant |M_n| < r, \\ S[\dots r+3) \parallel 1\overline{11} \parallel S[r+6\dots), & \text{if } |M_n| = r, \end{cases} \\ S \leftarrow F(S), \\ C_i \leftarrow S[\dots |M_i|] \oplus M_i, \\ S \leftarrow M_n \parallel 10^{r-|M_n|} \parallel \overline{11} \parallel S[r+3\dots). \end{gathered}$$

Before the operation is called key must be loaded.

Ciphertext Processing. The operation has ciphertext $C \in \mathbf{T}^*$ split into blocks $C_1, \dots, C_n$ as input arguments and plaintext $M \in \mathbf{T}^*$ split into blocks $M_1, \dots, M_n$ such that $|M_i| = |C_i|$ as output arguments.

Each intermediate block C_i is processed as follows:

$$\begin{gathered} S \leftarrow S[\dots r+3) \parallel 11\overline{1} \parallel S[r+6\dots), \\ S \leftarrow F(S), \\ M_i \leftarrow S[\dots |M_i|) \ominus C_i, \\ S \leftarrow M_i \parallel 11\overline{1} \parallel S[r+3\dots). \end{gathered}$$

The last block C_n is processed as follows:

$$\begin{gathered} S \leftarrow \begin{cases} S[\dots r+3) \parallel 0\overline{11} \parallel S[r+6\dots), & \text{if } 0 \leqslant |C_n| < r, \\ S[\dots r+3) \parallel 1\overline{11} \parallel S[r+6\dots), & \text{if } |C_n| = r, \end{cases} \\ S \leftarrow F(S), \\ M_n \leftarrow S[\dots |C_n|) \ominus C_n, \\ S \leftarrow M_n \parallel 10^{r-|C_n|} \parallel \overline{11} \parallel S[r+3\dots). \end{gathered}$$

This operation is an inverse to the plaintext processing operation.

Tag Unloading. The operation has no input arguments and has tag $T \in \mathbf{T}^*$ split into blocks $T_1, \dots, T_n$ as output argument.

Each intermediate block T_i is generated as follows:

$$\begin{gathered} S \leftarrow S[\dots r+3) \parallel \overline{1}1\overline{1} \parallel S[r+6\dots), \\ S \leftarrow F(S), \\ T_i \leftarrow S[\dots r), \\ S \leftarrow 0^r \parallel \overline{1}1\overline{1} \parallel S[r+3\dots). \end{gathered}$$

The last block T_n is generated as follows:

$$S \leftarrow S[\dots r+3) \parallel \overline{111} \parallel S[r+6\dots),$$
$$S \leftarrow F(S),$$
$$T_n \leftarrow S[\dots |T_n|),$$
$$S \leftarrow \begin{cases} 0^{|T_n|}\|1\|0^{r-|T_n|-1} \parallel \overline{111} \parallel S[r+3\dots), & \text{if } 0 \leqslant |T_n| < r, \\ 0^r \parallel \overline{111} \parallel S[r+3\dots), & \text{if } |T_n| = r. \end{cases}$$

Before the operation is called key must be loaded and associated data and/or plaintext (ciphertext) processed.

Hash Unloading. The operation has no input arguments and has hash value $H \in \mathbf{T}^*$ split into blocks $H_1, \dots, H_n$ as output argument.

Each intermediate block H_i is generated as follows:

$$S \leftarrow S[\dots r+3) \parallel \overline{1}10 \parallel S[r+6\dots),$$
$$S \leftarrow F(S),$$
$$H_i \leftarrow S[\dots r),$$
$$S \leftarrow 0^r \parallel \overline{1}10 \parallel S[r+3\dots).$$

The last block H_n is generated as follows:

$$S \leftarrow S[\dots r+3) \parallel \overline{11}0 \parallel S[r+6\dots),$$
$$S \leftarrow F(S),$$
$$H_n \leftarrow S[\dots |H_n|),$$
$$S \leftarrow \begin{cases} 0^{|H_n|}\|1\|0^{r-|H_n|-1} \parallel \overline{11}0 \parallel S[r+3\dots), & \text{if } 0 \leqslant |H_n| < r, \\ 0^r \parallel \overline{11}0 \parallel S[r+3\dots), & \text{if } |H_n| = r. \end{cases}$$

Before the operation is called associated data must be processed.

Pseudorandom Number Generation. The operation has no input arguments and has pseudorandom number $R \in \mathbf{T}^*$ as output argument split into blocks $R_1, \dots, R_n$ as output argument.

Each intermediate block R_i is generated as follows:

$$S \leftarrow S[\dots r+3) \parallel \overline{1}11 \parallel S[r+6\dots),$$
$$S \leftarrow F(S),$$
$$R_i \leftarrow S[\dots r),$$
$$S \leftarrow 0^r \parallel \overline{1}11 \parallel S[r+3\dots).$$

The last block R_n is generated as follows:

$$S \leftarrow S[\dots r+3) \parallel \overline{11}1 \parallel S[r+6\dots),$$
$$S \leftarrow F(S),$$
$$R_n \leftarrow S[\dots |R_n|),$$
$$S \leftarrow \begin{cases} 0^{|R_n|}\|1\|0^{r-|R_n|-1} \parallel \overline{11}1 \parallel S[r+3\dots), & \text{if } 0 \leqslant |R_n| < r, \\ 0^r \parallel \overline{11}1 \parallel S[r+3\dots), & \text{if } |R_n| = r. \end{cases}$$

Before the operation is called key must be loaded.

2.9 Sponge and Spongos

The AE algorithms of the prevoious section are presented in Specficaition in the form of the `Sponge` layer. The layer maintains 6 commands:

1) `Absorb[DATA]`(X) — process public input data $X \in \mathbf{T}^*$;

2) `Absorb[KEY]`(X) — process secret input data $X \in \mathbf{T}^*$;

3) `Squeeze[PRN]`(n) — generate n pseudorandom trits;

4) `Squeeze[MAC]`(n) — generate MAC of n trits;

5) `Encr`(X) — encrypt plaintext $X \in \mathbf{T}^*$;

6) `Decr`(Y) — decrypt ciphertext $Y \in \mathbf{T}^*$.

All these commands change the sponge state. Additionally, the last 4 commands return output data.

Let us interpret the `Sponge` layer as an *automaton*, which maintains a state S and this state is changed when operations are processed. The automaton starts in a prescribed state (actually, zero), each its new state is fully determined by a sequence of previously processed commands and their inputs. We call this sequence a *program*.

Let us remind that a sponge state S is divided into the rate part, the control part (two control trytes) and the capacity part. It would be convenient to denote the capacity part of S by $S[C]$. Here C is the range of capacity trit indices.

Let two automatons run programs P_1 and P_2. Let $S_{1,0}, S_{1,1}, \ldots, S_{1,t}$ and $S_{2,0}, S_{2,1}, \ldots, S_{2,\tau}$ be the sequences of their states induced by the programs. Note that $S_{1,0} = S_{2,0} = 0^{729}$.

Definition 2.1. *Two automatons* collide *if the capacity parts of their states coincide after running some programs:* $S_{1,t}[C] = S'_{2,\tau}[C]$.

After a collision, the automatons are indistinguishable: they will process the same commands in the same way. Such indistinguishability is unacceptable from the security point of view. Indeed, we cannot allow, for example, that after absorbing different data the automatons will squeeze the same MACs.

Trivially, automatons collide if $P_1 = P_2$. If $P_1 \neq P_2$, then there are two reasons for a collision:

1) two different commands in P_1 and P_2 yield the same sponge states $S_{1,i}$ and $S_{2,j}$, $i \leqslant t$, $j \leqslant \tau$. This situation is also unacceptable from the security point of view;

2) an *internal collision* occurs: $S_{1,i} \neq S_{2,j}$ but $F(S_{1,i})[C] = F(S_{2,j})[C]$ for some $i < t$ and $j < \tau$. As we have seen before, the main task in designing a sponge automaton is to prevent internal collisions. So we can assume that an adversary can manage an internal collision with only negligible probability.

Theorem 2.1. *Let two* `Sponge` *automatons be collide after running programs* P_1 *and* P_2. *Then either* $P_1 = P_2$ *or an internal collision occurs.*

Proof. Let $S_{1,t}[C) = S'_{2,\tau}[C)$ be a target collision.

If $t = 1$ and $\tau > 1$, then we get an internal collision. Indeed, $S_{1,0} \neq S_{2,\tau-1}$ since the trit $c[1]$ in the first control tryte of $S_{1,0}$ is zero although the same trit in $S_{2,\tau-1}$ is not (we use the notations of Specification). The case when $t > 1$ and $\tau = 1$ is treated similarly.

Let us suppose now that $t > 1$ and $\tau > 1$. If $S_{1,t-1} \neq S_{2,\tau-1}$, then we get an internal collision: $F(S_{1,t-1})[C) = F(S_{2,\tau-1})[C)$. If $S_{1,t-1} = S_{2,\tau-1}$, we can apply the above arguments recursively. As a result, in the absence of internal collisions, it must hold that $t = \tau$ and $S_{1,i} = S_{2,i}$ for all $i = 0, 1, \ldots, t$.

Now it is sufficient to prove that $P_1 = P_2$. It is indeed the fact because the types of commands and their arguments are unambiguously described in the rate and control parts of sponge states. More precisely, let us consider the first difference between P_1 and P_2.

1. If the commands which differ have distinct types, then control trits $c[0]c[2]$ in corresponding sponge states will be distinct.

2. If the commands which difer process ternary strings of different lengths, then control trits $c[0]c[2]$ will be distinct.

3. If the commands which differ process different ternary strings of the same length, then the rate parts will be distinct.

□

Remark 2.1. *In fact, the* `Sponge` *automaton gives even greater guarantees of protection against collisions than those mentioned in the theorem. It is guaranteed not only the fact that different commands will lead to a difference in the rate and control parts of the state. It is guaranteed that the parts will begin to differ immediately after the difference in the commands. This functionality is achieved by taking into account the control trytes of both the current and the next commands.*

So in the `Sponge` automaton, we allow that neither the next command nor the length of the currently processed ternary string is not known at the beginning of processing. These settings are very universal and flexible. But to achieve security guarantees in the settings, we forced to make the processing logic enough complicated: we have to control codes of the current and next commands, the length of the processing data block, the flag that this is the last block of the command. This somewhat reduces the effectiveness of cryptoprocessing.

An even more serious decrease in efficiency is that we need to call the sponge function F at the end of each command. We call such a call a *commit*: we indeed commit changes induced by the current command before moving on to the next command. As a result, if commands operate with small portions of data (high fragmentation that exactly happens in MAM2), efficiency decreases very dramatically.

There exists a simple solution to the fragmentation problem if we move on to the processing of strictly formatted data. The data format is defined in some external specification up to local data attributes (options, lengths, numbers of repetitions) that encoded before referenced data fields. As a result, the current and previous data trits in the encoded stream completely determine the semantics of the next trit. Here under the semantics we understand which command should be applied to the data.

If we absorb data attributes in a sponge state before referenced data fields (which was not always the case in `Sponge`), then the automaton can be significantly simplified while saving the security guarantees of the above theorem.

We call the new automaton `Spongos` (from $\sigma\pi o\gamma\gamma o\varsigma$, "sponge" in Greek) meaning the new "language" of working with the sponge function.

Let us describe how `Spongos` works.

1. A message is encoded to a string of trits. The string consists of encoding descriptors which precede fields values. All lengths are known. The only uncertainty are to-encrypt, to-decrypt, and to-squeeze values. All trits of these values are marked with the corresponding labels. Other trits, including trits of encoding descriptors, are marked with the to-absorb label.

2. Trits of the stream may additionally have the to-commit label. This label is assigned to the trits in accordance with the pre-defined data format.

3. A sponge automaton maintains a state S (initially zero) and an offset pos (initially zero). The offset is incrementally increased and reset to 0 when it reaches 486 (the margin of the rate part). When the offset is reset, the state is transformed using a sponge function F: $S \leftarrow F(S)$ (commit).

4. The automaton processes sequential trits of the encoded string. Let t be the current trit. Then:

   1) if t is marked with the to-commit label, then $S \leftarrow F(S)$ (commit), $pos \leftarrow 0$;

   2) if t is marked with the to-absorb label, then $S[pos] \leftarrow t$;

   3) if t is marked with the to-squeeze label, then $t \leftarrow S[pos]$, $S[pos] \leftarrow 0$;

   4) if t is marked with the to-encrypt label, then $(t, S[pos]) \leftarrow (t \oplus S[pos], t)$;

   5) if t is marked with the to-decrypt label, then $t \leftarrow t \ominus S[pos]$, $S[pos] \leftarrow t$;

   6) if t is marked with the to-absorb, to-squeeze, to-encrypt or to-decrypt labels, then $pos \leftarrow pos + 1$. If pos becomes equal to 486, then $S \leftarrow F(S)$ (commit), $pos \leftarrow 0$.

5. Return the post-processed string of trits.

The main idea here is that we significantly decrease the number of commits, that is, the number of fragments from the automation point of view. First, we gather all neighboring to-absorb trits in large fragments. Second, we get rid of the commits at the switchings to-absorb $\rightarrow$ to-squeeze, to-absorb $\rightarrow$ to-encrypt, to-absorb $\rightarrow$ to-decrypt, etc. Some excluded commits remain necessary from the security point of view (for example, a commit after absorbing a key which activates this key before processing next trits). Therefore, we leave the possibility of forced commits.

The price for decreasing the number of commits is that we need to absorb encoding descriptors. But this price is not as great as the efficiency gain.

Theorem 2.2. *Let two `Spongos` automations be collide after running programs P_1 and P_2. Then either $P_1 = P_2$ or an internal collision occurs.*

Proof. Let us save the notations of the previous proof and repeat its main arguments.

If internal collisions do not occur, then $t = \tau$ and $S_{1,i} = S_{2,i}$ for all $i = 0, 1, \ldots, t$. This is only possible with $P_1 = P_2$. Indeed, if $P_1 \neq P_2$, then in a ternary string constructed at Step 1

of the algorithm described above, some of the absorbed trits must be different. Otherwise, the format is not strict and the semantics of the next trit is not completely determined by the previous ones. □

Chapter 3

Digital Signature Algorithms

3.1 Concept

Digital signature is an essential cryptographic primitive used in IOTA/MAM. It provides authenticity IOTA transaction, and authenticity of messages in MAM channel.

Definition 3.1 (Digital signature scheme)**.** *Formally, digital signature scheme is a triplet of algorithms (`KeyGen`, `Sign`, `Verify`), where:*

- `KeyGen` $: 1^k \mapsto (pk, sk)$ *is a probabilistic algorithm, which takes as input a security parameter k (in unary notation), and returns secret and public keys (from some secret key space, and public key space);*
- `Sign` $: (sk, X) \mapsto \sigma$ *is a potentially probabilistic algorithm, which takes as input a message X (from some message space) and generates signature σ (from some signature space);*
- `Verify` $: (pk, X, \sigma) \mapsto b$, $b \in \{0, 1\}$ *is a deterministic algorithm, which verifies signature for a message upon a public key.*

The following property (correctness) is required:

- *if (pk, sk) is a result of `KeyGen` invocation, and σ is a result of `Sign` invocation with parameters (pk, sk) and some message X from message space, then* `Verify`$(pk, X, \sigma) = 1$.

Besides correctness, the scheme needs to be secure. This is formalized by the notion of *Existential Unforgeability Under Chosen Messages Attack* (EU-CMA).

Definition 3.2 (EU-CMA)**.** *The signature scheme (`KeyGen`, `Sign`, `Verify`) satisfies EU-CMA, if for any adversary A it's hard (in a sense of computation time and probability of success) to win in EU-CMA game, defined below.*

Here we use notation "$A \Rightarrow B\colon X$" meaning party A sends message X to party B, "$A \Leftarrow\colon X$" meaning party A receives X, and "$A \Rightarrow\colon X$" meaning party A outputs X.

Game EU-CMA

Conditions: Adversary A plays a game with signing oracle S^{sk}, which has access to a secret key sk. A, which has access only to a public key pk, is allowed to ask S^{sk} for signatures of several messages, chosen by A. The oracle queries may be adaptive, that is, a message may depend on

the oracles answers to previously queried messages. S^{sk} is obligated to return correct signatures (she implements `Sign` algorithm for that). After that, A wins if she can produce a signature for some new message.

Steps:

1. Key generation (S^{sk} obtains access to sk, A gets only pk):
   (a) $(sk, pk) \leftarrow \texttt{KeyGen}(1^k)$;
   (b) $A \Leftarrow: pk$;
   (c) $S^{sk} \Leftarrow: (pk, sk)$.
2. Adversary A makes several (potentially adaptive) requests to oracle S^{sk} for valid signatures:
   (a) $A \Rightarrow S^{sk}: X_i$;
   (b) $S^{sk} \Rightarrow A: \sigma_i$, so that $\texttt{Verify}(pk, X_i, \sigma_i) = 1$.
3. As a result, A produces a valid signature σ' for some messsage X', not appeared in the preceding requests to oracle:
   (a) $A \Rightarrow: (\sigma', X')$;
   (b) A wins if $\texttt{Verify}(pk, X', \sigma') = 1$ and $\forall i\, X' \neq X_i$.

Particular digital signature scheme is characterized by *capacity* C, which is defined as a maximum number of different messages which could be signed by a single key pair, so that scheme stays secure. For some schemes, capacity is (practically) unlimited. But there're schemes (e.g. hash-based) of restricted capacity. We define:

- *one-time signature schemes*(OTS), when only one message signature is allowed ($C = 1$);
- *few-time signature schemes*(FTS), which supports some small number (typically a dozen) of messages ($C > 1$);
- *many-time signature schemes*(MTS), when a lot of messages are supported, but it's still restricted; By choosing proper parameters, we usually can achieve suitable capacity, e.g. $C = 2^{40}$.

For OTS, we should use a special case of EU-CMA, which we call EU-CMA-1. EU-CMA-1 is characterized by that on the step 2 only one request X to oracle is allowed.

Similarly, for MTS with capacity C, we can use notion of EU-CMA-C, which means that only up to C requests to signing oracle are allowed.

Classical digital signature scheme definition, given above, corresponds to *stateless* signatures. However, some schemes (especially hash-based) are *stateful*, which means that signer must maintain a state between `Sign` invocations. We can formalize it by a slight modification of `Sign` definition:

- $\texttt{Sign} : (sk, X) \mapsto (sk', \sigma)$ is a potentially probabilistic algorithm, which takes as input a message X and returns signature σ with modified secret key sk', which will be used as secret key for the next use.

So, for stateful signature, secret key is modified after each `Sign` invocation. All other definitions and security notions remain the same.

3.2 Hash-based signatures

3.2.1 Overview

Lamport signature. Hash-based cryptography began its development from Lamport signature scheme [72]. The idea is having a one-way (hash) function $h : \mathbf{A}^n \to \mathbf{A}^n$, we can construct a scheme for signing single-bit message: the secret key consists of a pair of random strings $sk_0, sk_1 \in \mathbf{A}^n$, and public key is calculated as their hash-images: $pk_0 = h(sk_0), pk_1 = h(sk_1)$. Then, for signing of bit 0, signer reveals sk_0, and otherwise, for signing of 1, he reveals sk_1. This idea could be simply extended to sign messages of arbitrary length. At first, additional (or the same) collision-resistant hash function $H : \mathbf{A}^* \to \mathbf{A}^L$ should be employed, which is used to compress messages and create message digest of fixed length L (so-called hash-then-sign approach). Then, to sign L bits of message digest, we should extend secret key sk to a sequence of L pairs of random strings — $(sk_0^1, sk_1^1), (sk_0^2, sk_1^2), \ldots, (sk_0^L, sk_1^L)$, and public key is calculated in the same way described above (as hash-images of secret values). For each bit of message digest we should reveal a specific preimage of the corresponding pair.

Then, to make secret key compact, we can generate all $2L$ strings from a single seed string, having big enough entropy. So signer, instead of storing $2L$ strings each of length n, stores a value $seed$ of length n, and, during signing, generates $sk = \texttt{KDF}(seed)$, where `KDF` means some key-derivation function.

Also, we can compress public key by using some hash function f and publishing hash-value of pk_j^i as a public key: $pk = f(pk_0^1||pk_1^1||...||pk_0^L||pk_1^L)$.

This scheme is reasonably efficient: it has short public and secret key, fast generation and verification time. But it still has a drawback — large signature size (L strings each of length n). Such situation is typical for almost all hash-based signature schemes.

Another drawback is that this scheme is one-time signature (OTS) (see 3.1). Signer can use it for signing only one message. The reason is that he reveals part of the secret key in a signature value (more exactly half of the key). Publishing the second signature will reveal greater part of the key, enough to make a forgery.

Winternitz OTS. The idea of Winternitz improvement over Lamport scheme is to use w-length chains of hash evaluations (instead of single hash evaluation). And, in order to bind signature to a message digest M, one can use not only position in a pk, but also a depth in the hash-chain. Winternitz one-time signature scheme (WOTS) is parametrized by a value w, which defines a trade-off between signature size and generation/verification time: compared to a Lamport scheme, signature size is compressed (roughly) by a factor of $\log_{|\mathbf{A}|} w$, while generation/verification time is increased in w times. WOTS is described in detail in 3.2.2.

Graph-based OTS. In [40], a generalization of hash-based OTS is proposed, including WOTS and Lamport, called graph-based OTS (GBOTS). The idea is that we can use arbitrary directed acyclic graph of hash-evaluations: secret key is a subset of vertices, each edge means a hash-evaluation, and public key is another subset of vertices (built from a secret-key vertices via specific hash-evaluations in accordance to edges of the graph). The signature is another subset of vertices, that lies (in some sense) between pk and sk vertices, and defined by a message digest M. It's easy to see, that WOTS scheme is a particular case of GBOTS, where graph is a set of chains of length w.

Bleichenbacher and Maurer studied the efficiency of different graphs. The main property to maximize is a size of message space (number of different messages that could be supported by the graph) with respect to a number of vertices in the graph, or with respect to the signature size. In [41] it's shown that WOTS is asymptotically maximally efficient with respect to the signature size. However, there are another graphs proposed, which have some advantages in efficiency (e.g. not asymptotically, but locally).

Subsets-based schemes. Another idea to improve Lamport scheme is based on the following observation. In Lamport scheme, we use $2L$ secret values to support signatures for 2^L (or, generally, $|\mathbf{A}|^L$) messages. However, we can support more messages, if we employ more subsets of $2L$ values to represent messages. Actually, we have $\binom{2L}{L}$ (number of combinations) subsets of size L, such that no subset contains another subset of the family (this property is required to prevent a straightforward forgery). Then, the signature is pre-images of members of a subset defined by the message.

For this scheme to work, we need a safe mapping from message space to subsets of length L (so-called subset selection). Several constructions for subset-selection are presented (e.g. in [44]). The main disadvantage of these schemes is not satisfactory time needed for a safe subset-selection.

Although, this optimal efficiency (support $\binom{2L}{L}$ messages on signature size L) is difficult to achieve, we can use smaller subsets. In [168], Reyzin and Reyzin proposed a scheme, which is a development of the idea described above: to use subsets of size L, where L is much less then total number of secret values. This allows to reduce signature size with respect to secret key, and allows to use simple subset-selection: just evaluating of hash function $H(X)$, splitting hash-string into L chunks, each of them pointing (by index) to one of $2L$ values. The subset constructed in this way may contain duplicate (redundant) values, but with some controlled probability, so it's argued that this is not a problem. This technique is called Hash to Obtain a Random Subset (HORS).

More interesting, that this scheme is not one-time, but few-time (FTS). This means that one can produce several signatures, while security decreases gradually with a number of signatures issued. In this case (FTS), security of HORS is based on a new special problem upon hash function H called *subset-resilience problem*. Informally, it can be formulated as finding a message X' so that the corresponding subset (specified by $H(X')$) is fully included into some of the subsets corresponding to a set of chosen messages $(X_1, X_2, \ldots, X_N)$. This problem directly corresponds to producing a forgery during chosen-messages attack (EU-CMA game).

The hardness of subset-resilience problem could be argued, but it's not unmistakable. For example, adaptive attacks are proposed in [13].

Many-time signatures. OTS and FTS are not practical for many applications.

The idea of the simplest MTS signature, *Merkle Signature Scheme* (MSS), is:

1. Use several instances of some OTS/FTS.
2. Combine their public keys into a binary hash tree, called a Merkle tree, so that the root of the tree becomes a public key of the MTS scheme.
3. For signing messages, use OTS/FTS instances successively. Provide authentication path from an OTS public key to root alongside with signature value.

Choosing appropriate tree height D you can support theoretically any capacity C of the scheme: $C = 2^D$. However in practice, it's still restricted by about 2^{20}: the reason is that key-generation time grows linearly with C, and it becomes inappropriate. MSS is described in detail in 3.2.3.

There is an approache for increasing capacity of tree-based MTS, called *hypertrees*. In CMSS [55], a combination of two layers of Merkle trees is proposed. On the first layer you have a single tree of height D_1, on the second — 2^{D_1} trees of height D_2. Then, the capacity of the scheme $C = 2^{D_1+D_2}$.

Each tree from the second layer is authenticated upon the first layer tree: a leaf of the first layer tree, which is actually an OTS instance, signs the root of corresponding tree from the second layer. The idea that you can afford generate (independently) two trees, each of 2^{20} capacity. Then, by using this scheme, you can support capacity up to 2^{40}. However, it comes at the expense of additional OTS signature value, alongside with authentication path, so the size of signature increases.

In GMSS [54], a generalization of this approach introduced. It's proposed to use several layers of trees, with different heights and different parameters (e.g. different w in WOTS). Maximal efficiency is estimated, and optimal constructions are proposed.

Stateless signatures. While described above hypertree-based MTS schemes (especially GMSS) are able to provide suitable for practice capacity, they are still stateful (see 3.1): signer needs to remember what OTS instances have been already used (usually, he uses OTS instances successively, from the most left leaf to the most right one, so signer just need to remember current index). This is a significant drawback for some applications. For example, it's difficult to support concurrent (parallel) signings. However, by applying some improvement, we can achieve a stateless tree-based signature.

The main idea is that if total capacity of hypertree is big enough, signer can choose OTS instance randomly, ignoring insignificant probability of choosing already used one (so-called *index-collision*). Also, index could be chosen not randomly, but calculated from a message digest.

In SPHINCS [28], hypertree of 12 layers of trees of height 5 is used, which supports 2^{60} keys in total. Moreover, each key is a FTS (HORS) instance, which mitigates effect of index-collisions.

Also, advanced stateless schemes SPHINCS+ [118] and Gravity [14] take part in NIST Post-Quantum Crypto Competition (NPQCC). Generally, they are modifications of original SPHINCS design, with some local improvements and optimizations. For example, Gravity reduces signature size by more than half with respect to SPHINCS.

3.2.2 Generalization of WOTS

In this section, a generalization of WOTS is presented. The goal is to encompass novel approach introduced in IOTA called *normalization* (see 3.3.3.2).

Parameters. WOTS is parametrized by a number w, which defines the length of Winternitz chains (hereinafter — just chains). Essentially, parameter w defines a trade-off between signature size and speed (time for signing/verification). As w increases, signature size decreases(with

near-logarithmic speed), and generation/verification time grows (linearly). Popular values for w are 16 (on binary processors) and 27 (ternary processors). If $w = 1$, WOTS degenerates into Lamport scheme (described shortly in 3.2.1).

Without loss of generality, let's assume w is a power of $|\mathbf{A}|$. It will allow to use strings of length $\log_{|\mathbf{A}|} w$ to represent integers from 0 to $w - 1$.

Used primitives. WOTS is built on top of the following primitives:

- $H' : \mathbf{A}^* \times \mathcal{N} \to \mathcal{W}$ — a collision-resistant hash function, which maps message $X \in \mathbf{A}^*$ and nonce $N \in \mathcal{N}$ into an element of space $\mathcal{W} = \{0, 1, \ldots, w-1\}^L$;
- $\mathcal{W}' \subset \mathcal{W}$ — a subset of $\mathcal{W}$, which we call the *target subset*; we require the following properties for $\mathcal{W}'$:
  - *verifiability*, which means that it should be easy to check whether value $A \in \mathcal{W}$ lies in the target subset;
  - *intersectability* (analogue of Property 1), which means that for any $A^1, A^2 \in \mathcal{W}'$, there are positions where A^1 is greater than A^2, and there are positions where vice-versa, A^2 greater than A^1 (here A_i means i-th trit of A):

$$A^1 \neq A^2 \Rightarrow \exists i, j : A^1_i > A^2_i, A^1_j < A^2_j.$$

- $h : \mathbf{A}^{n_w} \to \mathbf{A}^{n_w}$ — an one-way function which is used to construct Winternitz chains (however, there are other options for constructing chains, this question is discussed in 3.2.5).

H' is required to be collision-resistant (in a sense that it's hard to find two pairs (X, N) and (X', N'), different at least in one coordinate, so that their images equal). At the same time, h is required to be only one-way. However, there are significant troubles in building security reduction to one-wayness property of h. This question is discussed in 3.2.5.

Secret and public keys. In general, secret key sk consists of L random strings of length n_w:

$$sk = (sk_1, sk_2, \ldots, sk_L), \quad sk_i \in \mathbf{A}^{n_w} \forall i \in \{1, 2, \ldots, L\},$$

where value of L depends on the used H'.

Easy to see, that for practical parameters (where L is large enough to support CR of H), sk is pretty large. So in practice, to minimize a secret value to store, one can generate sk from a short seed value in a pseudorandom fashion. Moreover, when WOTS is used as a building block of MSS-like scheme (3.2.3), secret keys for all WOTS instances should be generated from a single seed value. This question is discussed in 3.2.4.

Signature generation. Signature $\sigma \in \mathcal{N} \times \mathbf{A}^{n_w L}$ of a message $X \in \mathbf{A}^*$ on a secret key $sk \in \mathbf{A}^{n_w L}$, is evaluated non-deterministically via the following steps:

1. Nonce $N \in \mathcal{N}$ is chosen randomly, and message digest is calculated: $A = H'(X, N)$.
2. If $A \in \mathcal{W}'$ — go to the next step. Terminate execution otherwise. The membership test is allowed thanks to verifiability of $\mathcal{W}'$.

3. Each number A_i defines a position in a hash-chain i, built from sk_i. More formally,

$$\sigma_i = h^{A_i}(sk_i), \quad i = 1, 2, \ldots, L.$$

4. Resulting signature is:

$$\sigma = (N, \sigma_1, \sigma_2, \ldots, \sigma_L).$$

Verification. Verification consists of the following steps:

1. Message digest is calculated from message and nonce: $A = H'(X, N)$.
2. If $A \in \mathcal{W}'$ — go to the next step. Otherwise — return 0. The membership test is allowed thanks to verifiability of $\mathcal{W}'$.
3. Recover key $pk' = (pk'_1, pk'_2, \ldots, pk'_L)$ from the signature value:

$$pk'_i = h^{w-1-A_i}(\sigma_i), \quad i = 1, 2, \ldots, L.$$

4. Match pk' to pk: if $pk' = pk$, then return 1, otherwise — return 0.

Analysis of the scheme. In general, `Sign` is not deterministic, and could fail with some probability on step 2 in case if A doesn't hit into the target subset $\mathcal{W}'$. If H' is random (in some sense), probability of success $P = \frac{|\mathcal{W}'|}{|\mathcal{W}|}$. Signer can retry `Sign` execution about $\frac{1}{P}$ times in order to find appropriate nonce N so that step 2. will be successful. For some applications, this trait could be regarded as a useful feature (so-called *proof of work*).

The scheme is correct. Easy to see that step 2. of `Verify` doesn't break correctness of the scheme: if σ is indeed a result of honest `Sign` invocation, then nonce N and message M are suitable to each other (their image lies in the target subset).

Security of the scheme could be argued via the following sketch of a proof:

1. Suppose that scheme is broken in EU-CMA-1 game, and forgery (X', σ') is found based only on correct pair (X, σ).
2. The first option is that corresponding digests A and A' equal. Then, collision of H' is found, which violates requirements for H'.
3. The second option is, otherwise, $A \neq A'$. By the intersectability of $\mathcal{W}'$, there is a position i where A' is less than A. This means that to make a forgery, attacker had to invert (in some sense) a chain i deeper than it's inverted in (X, σ). This violates OW or CR properties of h (see 3.2.5 for detailed reduction).

Let's clarify requirements for h. There is a security proof that WOTS is EU-CMA-1, if h is one-way (OW) and collision-resistant (CR). There are modifications of original scheme, WOTS+ and WOTS-PRF, which allow to relax rquirements for h from CR to SPR. These schemes are discussed in 3.2.5.

At the same time, there are no practical attacks against WOTS found, that could employ finding of collisions in h. So, original WOTS is presumably secure even if h is only SPR. We're reasoning this in 3.5.

It worth to note, that the required property of h (SPR or CR) is an important question which also determines efficiency of the scheme: CR hash functions have significantly larger output size, and therefore, signature size of WOTS based on CR function is significantly larger. Security and sizes of SPR/CR functions (in Post-Quantum Epoch) are discussed in 1.3.4.

Checksum instantiation. The classic instantiation for H' and $\mathcal{W}'$ is based on using of checksum.

Target subset is defined in the following way:

$$\mathcal{W}' = \{A : \sum_{i=1}^{L_1}(w-1-A_i) = \sum_{i=1}^{L_2}(|\mathbf{A}|^{i-1}A_{L_1+L_2})\}.$$

Numbers L_1, L_2, L are calculated in the following way:

$$L_1 = \frac{n_m}{\log_{|\mathbf{A}|} w}, \quad L_2 = \lfloor \log_w L_1(w-1) \rfloor + 1,$$

$$L = L_1 + L_2.$$

To build H', a conventional CR hash-function $H : \mathbf{A}^* \to \mathbf{A}^{n_m}$ is used, where $n_m = L_1 \log_{|\mathbf{A}|} w$. Function H' is calculated deterministically in the following way:

1. Message digest is calculated: $M = H(X)$.

2. Message digest $M \in \mathbf{A}^{n_m}$ is split into L_1 chunks of length $\log_{|\mathbf{A}|} w$:

   $$M \to (c_1, c_2, \ldots, c_{L_1}), \quad c_i \in \mathbf{A}^{\log_{|\mathbf{A}|} w},$$

   and each chunk represents a number between 0 and $w-1$:

   $$c_i \to A_i, \quad A_i \in \{0, 1, \ldots, w-1\}, \quad i = 1, 2, \ldots, L_1.$$

3. Checksum C is calculated as a sum of integers representing chunks:

   $$C = \sum_{i=1}^{L_1}(w-1-A_i),$$

   and is split into additional L_2 chunks of length $\log_{|\mathbf{A}|} w$:

   $$C \to (c_{L_1+1}, c_{L_1+2}, \ldots, c_L),$$

   which again represents intergers between 0 and $w-1$:

   $$c_i \to A_i, \quad A_i \in \{0, 1, \ldots, w-1\} \quad i = L_1+1, L_1+2, \ldots, L.$$

4. Combining chunks from message digest and checksum, we have a result:

   $$A = (A_1, A_2, \ldots, A_L).$$

The result of H' always hits into target subset. Therefore, `Sign` becomes deterministic. Also, nonce N is not used in the scheme: there are no need to look over appropriate nonce value for a message so that image hits into target subset. This allows to simplify signature value:

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_L).$$

Easy to see, that such H' and $\mathcal{W}'$ satisfy all required properties:

– H' is CR, because it's a bijection of images of H, which is CR;

– $\mathcal{W}'$ is verifiable (simple check);

– $\mathcal{W}'$ satisfies intersectability (by properties of checksum of length L_2 added).

Normalization instantiation. Normalization is a novel construction from IOTA. It's described in detail in 3.3.3.2. In this case, target subset consists of members for which sum of digits equals to $T = \lfloor \frac{L(w-1)}{2} \rfloor$. More formally:

$$\mathcal{W}' = \{A : \sum_{i=1}^{L} A_i = T\}.$$

Easy to see that $\mathcal{W}'$ satisfies verifiability and intersectability.

H' is constructed on top of conventional CR hash-function $H : \mathbf{A}^* \to \mathbf{A}^{n_m}$ in the following way:

$$H'(X, N) = \langle H(X \parallel N) \rangle_w,$$

where $\langle x \rangle_w$ for a string $x \in \mathbf{A}^l$ denotes a splitting x into chunks of length $\log_{|\mathbf{A}|} w$ and each chunk representing an integer from 0 to $w - 1$.

Nonce space consists of strings of some fixed length n_n: $\mathcal{N} = \mathbf{A}^{n_n}$. Nonce length n_n is required to be large enough so that appropriate nonce for any message will be found. Easy to see, that H' is CR (collision in H' is automatically a collision in H). So, all required properties are satisfied.

Despite the fact that normalization-based WOTS scheme is correct and secure, it has a specific property not suitable for all applications: non-deterministic signature generation. For example, in 3.3.3.2 we show that signer needs about 176 retries (for $n_m = 243$ and $\mathbf{A} = \mathbf{T}$) in order to hit into target subset and generate a signature. For some applications, it could be even beneficial proof of work. Moreover, we can tune the difficulty by choosing target sum T.

Target collision resistance. Hash function H' could be viewed as a family of keyed hash functions, where nonces play a role of keys:

$$\mathcal{H}' = \{H'_N, N \in \mathcal{N}\}.$$

From this point of view, we can try to relax requirements: it's sufficient for the family $\mathcal{H}'$ to be *target collision resistant (TCR)* so that WOTS scheme will be secure.

The concept of TCR is introduced in [24]. The question of whether normalizaion produces a TCR family requires further analysis. In case of $\mathcal{H}'$ is TCR, it will allow to reduce the signature size significanlty. The reason is that birthday attack is not applicable, and hash size (L) could be shortened.

3.2.3 The Merkle signature scheme

MSS is a way to transform some OTS or FTS into many-time signature (MTS). So, depending on a specific OTS scheme, we could use more precise name like MSS[WOTS], MSS[WOTS+], MSS[HORS].

The idea is to combine 2^D public keys of OTS into a Merkle tree of height D, and use root of the tree as a single public key. Capacity of the scheme is 2^D.

Parameters. Parameter $D \in \mathbb{N}, D \geqslant 2$ defines a height of the tree. So D defines a tradeoff between capacity C of the scheme and key-generation time. By increasing D, C increases (exponentially), but signature size increases (linearly), signature generation/verification increases (linearly, if advanced tree-traversal algorithms will be used), and, the most problematic point, key-generation time increases (exponentially). The reason for hard key-generation is that one need to generate all C keys at once, while key-generation of separate OTS scheme (like WOTS) is usually hard too. Approaches for overcoming this key-generation difficulties and increasing capacity of tree-based schemes, like hypertrees, are discussed earlier in 3.2.1.

Hash function $h_T : \mathbf{A}^{2n_t} \mapsto \mathbf{A}^{n_t}$ is employed to build a tree. Function h_T is required to be OW. However, despite the fact that no attacks based on finding collisions of h_T is found, only security reduction constructed currently is to CR hash function. This question is discussed in 3.2.5.

To incorporate OTS public key into a tree as a leaf, some additional hash function $h_t^* \colon \mathbf{A}^* \mapsto \mathbf{A}^{n_t}$ is employed. In some shemes, to support security proof, advanced constructions of h_t^* are used (e.g. in XMSS, combining chunks of WOTS public key into L-tree).

Another parameter is a chosen OTS scheme.

Key generation. The signer generates C OTS keys-pairs. Digests of public keys $h_t^*(pk^{OTS_i})$ become leaves of Merkle tree. The inner nodes of the Merkle tree are computed according to the following construction rule: a parent node is the hash value of the concatenation of its left and right children. Public key of the scheme is a root of the tree, denoted as *root*.

Secret key consists of C OTS secret keys: $sk^{OTS_i}, i = 1, 2, \ldots, C$. But in practice, secret key is generated in a pseudorandom fashion from a single value *seed*. This is disscussed in 3.2.4.

During key-generation, in order to compute *root*, one need to build C OTS keys. But it's not necessary to store the full hash tree. There is an algorithm called `Treehash` (see in 3.2.3), which allows to make it more efficient. The idea is to successively compute leaves and, whenever possible, compute their parents.

Signature generation. MSS[OTS] signature consists of index i, OTS signature σ^{OTS_i} and authentication path $Auth = (Auth_1, ..., Auth_D)$:

$$\sigma = (i, \sigma^{OTS_i}, Auth).$$

Signer must be sure, that each OTS is used only once. This means that signer maintains a state, where information about already used keys is stored (scheme is stateful). The basic approach for this is that signer uses OTS successively, from the most left leaf to the most right.

Additionally, this approach makes possible some optimizations for signature generation time, by using effective so-called tree-traversal algorithms.

Signature verification. Here we suggest that, as in case of most OTS hash-based signatures, OTS public key pk_σ^{OTS} can be built from a signature σ, and verification of OTS is actually a matter of matching pk_σ^{OTS} to a known pk^{OTS}.

Verification consists of two steps:

1. Build OTS public key $pk_\sigma^{OTS_i}$ from signature σ^{OTS_i} of message X.
2. Verifying authentication data $(i, Auth)$ upon public key $root$ and OTS public key $pk_\sigma^{OTS_i}$.

Tree traversal techniques. One of the most important troubles in MSS is a signature generation time. Tree traversal techniques allow to do generation efficiently, and to choose a trade-off between memory consumption and time.

During a signature generation, we need to build authentication path. There are two extreme options to support this. One option is to store the whole tree (all nodes) and keys. In this case, time of a signature generation is minimal and memory consumption is maximal. The other option is to generate all keys and Merkle tree every time when the signature is created from a seed value. In this case, time of the signature generation is maximal and space is minimal. The idea of advanced tree-traversal algorithms to pre-compute and store some part of the tree between successive signature generations, so that some reasonable trade-off between time and memory is achieved.

All tree traversal techniques described here are based on efficient and easy root computation algorithm that is called `Treehash`. It allows to minimize memory consumption required for generating tree. Maximal memory that is required for the root computation is D nodes (hash values). This algorithm uses the data structure called stack. Stack provides two interfaces: `Stack.Push`(v) and `Stack.Pop`(). Every node and leaf in tree provides interface `Node.d`() that returns the height of this node. We will use these interfaces in algorithm description.

Moreover, `Treehash` is an efficient algorithm to build public key of MSS (as a root of the tree generated from the leaves).

Algorithm Treehash

Input:

- $D \in \{2, 3, \ldots\}$ (tree height).

Output:

- $r \in \mathbf{A}^{n_t}$ (the root of Merkle tree).

Steps

1. For $i = 0, 1 \ldots 2^{D-1}$ do:
   (a) $pk_i \leftarrow OTS_i$.`KeyGen`().
   (b) While (`Stack.Pop`().`d`() $= pk_i$.`d`()):

i. $node \leftarrow$ `Stack.Pop()`.

ii. $pk_i \leftarrow h_T(node||pk_i)$.

(c) `Stack.Push(`pk_i`)`.

2. Return `Stack.Pop()`.

There are three main Merkle tree traversal algorithms. We will give two characteristics for each algorithm: number of basic operations (time), and number of stored nodes/leaves (memory). Note that the basic operation is either hash evaluation or leaf value computation. Therefore all estimates that will be presented below are not exact for MSS that uses WOTS. The reason is that leaf computation in case of MSS[WOTS] is not a single hash evaluation (except for degenerate case when $w = 1$). But despite the fact that these estimates are a bit optimistic, they are a good reference point.

The *classic Merkle tree traversal* algorithm requires $2D$ basic operations per one signature and D^2 memory between signatures. After each signature it stores some new node/leaves values – one for each level.

The *logarithmic tree traversal* (also called *Szydlo tree traversal*) algorithm is based on the classic tree traversal but have slight improvements. This algorithm requires $2D$ basic operations per one signature and $3D$ memory between signatures.

The last algorithm is the *fractal tree traversal* algorithm. After each signature generation it stores some trees that are part of Merkle tree. These trees are updated after every signature generation. The algorithm is parametrized by the height d of the smaller tree. The algorithm requires less than $2D/d$ basic operations per one signature and less than $5D^2/2log_2D$ of memory between signatures.

The detailed descriptions of all these algorithms can be found in [26].

3.2.4 Key generation

For all OTS instances of the MTS scheme, secret key could be generated from a single *seed* in the following way:

1. $seed_i$ for each OTS instance is generated:

$$seed_i \leftarrow KDF(seed, i), \quad i = 1, 2, \ldots, C.$$

2. Secret key for each OTS instance is generated from the corresponding $seed_i$:

$$sk \leftarrow KDF^{OTS}(seed_i).$$

3.2.5 Relaxing requirements for hash functions

Overview. Construction of WOTS signature is based on two hash functions:

– H, which is used to transform message $X \in \mathbf{A}^*$ to a string of fixed length (message digest) $M \in \mathbf{A}^{n_m}$;

– h, which is used to construct a Winternitz chain for each of L blocks of message digest.

Security of signature scheme (EU-CMA-1) directly depends on properties of these functions.

Regarding H, things are pretty simple: we require second preimage resistance (SPR) and even collision-resistance (CR) of H.

However for h, there is a chance that we can relax requirements. Of course, it's still necessary for h to be OW: otherwise adversary A could just invert some Winternitz chain and break EU-CMA.

But CR seems like unnecessary property at first glance. At least, there are no attacks found based on solving $\texttt{Coll}[h]$. However, when it comes to provable security, only reduction to CR is constructed.

There are two positive effects from getting rid of CR requirement:

1. More reliable assumption: a scheme which security is based on easier problem ($\texttt{Coll}[h]$) has more chances to be broken in future than scheme based on more difficult problem (e.g. $\texttt{2Pre}[h]$, or $\texttt{Inv}[h]$). Moreover, signature scheme which is based on OW is maximally secure: there is a well-known result — OW functions exist if secure digital signature schemes exist [170]. This means that there is always a secure variant of such scheme, in case if any secure signature scheme exists.

2. Improved effectiveness: if rely on CR, because of generic attack based on birthday paradox, we need to increase (twice in pre-quantum world) the hash size for the same level of security. This leads to increasing of signature size twice (with the same WOTS parameters). In Post-Quantum Epoch, this difference is less (see 1.3.4), but still significant.

These reasons stimulated a development of more advanced constructions, which are proven to be secure based on weaker assumptions. In this section we describe two alternative solutions — WOTS+ and WOTS-PRF. The only difference between these schemes and original WOTS, is how Winternitz chains are constructed. Let's use the same notations and definitions which we use in 3.2.2. Additionally, let's call (intermediate) value in chain i in position j as $c_i^j \in \mathbf{A}^{n_w}$. Then, in original WOTS, chain is defined in the following way:

$$c_i^j = \begin{cases} sk_i, & \text{if } j = 0, \\ h(c_i^{j-1}), & \text{if } j \in \{1, 2, \ldots, w-1\}. \end{cases}$$

Another chain constructions are used in WOTS+ and WOTS-PRF (see later).

Then, let's call $F : \{0, 1, \ldots, w-1\} \times \mathbf{A}^{n_w} \mapsto \mathbf{A}^{n_w}$ a chaining function. F is defined in the following way: $F(j, sk_i) = c_i^j$, where c_i^j defined above. For original WOTS, we can write a simple representation of F: $F(j, sk_i) = h^j(sk_i)$, however for WOTS+ and WOTS-PRF representation is more complex, so we will define it through c_i^j in an iterative manner.

Original WOTS reduction. We will provide just a sketch of the proof here. For a complete proof, see [84].

Security proof is based on the following approach. We suppose that the scheme is not secure, which means that there is adversary algorithm A, which wins in EU-CMA-1 game (see 3.1). We will try to construct another algorithm B, which will have oracle access to A, and solves (in some restricted time, with some significant probability) some of hash function problems, e.g. $\texttt{Coll}[h]$. Algorithm B is allowed to impersonate signing oracle S^{sk} (from EU-CMA-1 game) to A.

We wil construct B which solves the composite problem $\texttt{Coll}[H] \vee \texttt{Inv}[h] \vee \texttt{Coll}[h]$ (it means that B will solve at least one of three problems — $\texttt{Coll}[H]$, $\texttt{Inv}[h]$, $\texttt{Coll}[h]$).

1. On input, algorithm B gets value $y \in \mathbf{A}^{n_w}$. She asked to find x so that $h(x) = y$, or to find collision in h, or collision in H.

2. B generates sk and pk of WOTS in a proper way. Then, she modifies pk: she chooses randomly a chain $\alpha \in \{1, 2, \ldots, L\}$ and a position $\beta \in \{1, 2, \ldots, w-1\}$. She puts challenge y into position (α, β), and builds chain from this position. Now, she publishes a modified pk, where $pk_\alpha = F(w - 1 - \beta, y)$. It worth to note, that now B is not able to sign every messages (she can't invert chain α deeper than $w - 1 - \beta$). So she hopes that A will not ask such unfortunate messages. However, she incorporated her challenge y into the WOTS instance.

3. Once A gets pk, she starts EU-CMA-1 game. B plays a role of signing oracle:

   (a) B receives signature request X from A. The digest $M = H(X)$ may or may not be fortunate for B: if $M_\alpha \geqslant \beta$, B returns a signature, otherwise it stops execution and loses the game. It could be shown that success probability is significant.

   (b) A receives signature for X, and produces a forgery (σ', X').

4. Once B receives a forgery (σ', X'), she tries to use it for solving one of her problems. At first, if $M' = M$ (where M' is a correspoding digest, $M' = H(X')$), then $\texttt{Coll}[H]$ is solved (collision — (M, M')). Otherwise, thanks to Property 1, $M'_i < M_i$ for at least some $i \in \{1, 2, \ldots, L\}$. In case it's not happened in chain α ($M'_\alpha \geqslant M_\alpha$), B stops execution and loses the game. Otherwise, which happens with probability at least $\frac{1}{L(w-1)}$, B has a chance to solve $\texttt{Inv}[h]$. Indeed, she incorporated y into position β, and now she has the deeper inversion of this chain: $x' = \sigma'_\alpha$ is a j-preimage of pk_α for some j. So we have $h^{j'}(y) = h^j(x') = pk_\alpha$, where $j' = w-1-\beta$, and $j' > j$. However, there is an undetermined fork here: hash-chain generated from x' may or may not go through y value. In the first case, which we call *enclosed chains*, B solves $\texttt{Inv}[h]$ with a solution $x = h^{j'-j-1}(x')$. In the second case, which we call *merging chains*, B solves only $\texttt{Coll}[h]$: indeed, since two hash-chains (generated from y and from x') must merge at some position (at least at the last position, in value pk_α), the collision at this position appears.

Easy to see, that actual barrier for getting rid of CR in the proof, is a case of merging chains. To overcome it, schemes WOTS+ and WOTS-PRF introduce additional mechanisms into chaining construction.

Interesting, that so-called merging chains case appears due to injective nature of hash functions. Cryptographic hash function could be seen as *public one-way injection*. If we could build *public one-way bijection (permutation)* for domain $\mathbf{A}^{n_w}$, we could employ it for WOTS, and security proof would be straightforward (merging chains case would be eliminated, as well as definition of $\texttt{Coll}[h]$ itself). However, currently there are no known efficient constructions, which would implement *public one-way permutation* (one can think about *trapdoor one-way permutation*, such as RSA scheme, but trapdoor one-way permutation is already a signature scheme itself).

WOTS+. WOTS+ scheme is a slight modification of WOTS, introduced in [119]. The idea is to use additional random values $(r_1, r_2, \ldots, r_{w-1}), r_i \in \mathbf{A}^{n_w}$, called masks, as part of the public key. Also, masks are used in modified chaining construction:

$$c_i^j = \begin{cases} sk_i, & \text{if } j = 0, \\ h(c_i^{j-1} \oplus r_j), & \text{if } j \in \{1, 2, \ldots, w-1\}. \end{cases}$$

Chaining function F changes accordingly.

This slight modification allows to build a reduction for a composite problem $\texttt{Coll}[H] \vee \texttt{Inv}[h] \vee \texttt{2Pre}[h]$.

Security proof is based on the same principles, as for original WOTS. Reduction scenario is modified in the following way:

1. On step 1, B gets, besides challenge y, another challenge $x_1 \in \mathbf{A}^{n_w}$. Instead of finding collision in h, B is asked to find x_2, such that $h(x_1) = h(x_2)$ (another words, to solve $\texttt{2Pre}[h]$).

2. On step 2, B generates keys (including random masks now). Then, besides incorporating y into some position (α, β), B incorporates (in some sense) x_1 into random position $\gamma \in \{\beta + 1, \ldots, w-1\}$ in the same chain α. Thanks to possibility of manipulating the masks, B is able to do this. Indeed, she can change mask r_γ in a way that x_1 will be an input for h at chain position γ (actually, $r_\gamma \leftarrow c_{\alpha,\gamma} \bigoplus x_1$). Chain is modified after position γ, and public key pk_α is modified too.

3. On step 4, case of merging chains is more fortunate for B: if chains merge at position γ, which appears with probability at least $\frac{1}{w-1}$, then B finds $x_2 \neq x_1$, such that $h(x_2) = h(x_1) = c_\alpha^\gamma$. So B solves $\texttt{2Pre}[h]$.

WOTS-PRF. WOTS-PRF is introduced in [53]. The problem of merging chains is solved in another way here.

The idea is to use (instead of hash-function h) a *family of pseudorandom functions (PRF)* F indexed by keys $k \in K$. Let's denote $f_k : \mathbf{A}^{n_w} \to \mathbf{A}^{n_w}$ a function from the family, chosen by key k: $f_k \in F, k \in K$.

Then, chain construction of WOTS-PRF is the following:

$$c_i^j = \begin{cases} sk_i, & \text{if } j = 0, \\ f_{c_i^{j-1}}(sk_i), & \text{otherwise.} \end{cases}$$

Then, in [53] arguments are given, that thanks to properties of PRF, number of so-called key-collisions is restricted, and therefore the case of merging chains appears only with some controlled probability.

However, it's shown in [134], that this proof is flawed, and nobody knows whether it could be fixed. So, despite that this scheme looks like pretty interesting alternative, security proof is absent for it.

SPR-MSS. The situation with MSS is pretty the same: we have a security reduction to only CR, while there are no known attacks based on finding collisions. To fix this, the same idea if using random masks are introduced in SPR-MSS [69] and XMSS [51].

Exactly, in XMSS[WOTS]:

1. Hash-lengths in Merkle tree and in WOTS equals: $n_w = n_t = n$. Also, despite loss of generality, we can assume $h_T = h$.

2. To construct a leaf of the tree from WOTS key, so called L-tree is used. L-tree is a not-complete unbalanced binary hash-tree, where L components of WOTS public key $(pk_1, pk_2, \ldots, pk_L)$ are leaves of the tree (since L is not a power of 2, it's not possible to build a complete Merkle tree). Then, roots of each L-tree serve as leaves of the main Merkle tree. So, we can call such a construction as *extended Merkle tree*. Extended Merkle tree consists of main Merkle tree, and 2^D L-trees. Total height of extended Merkle tree $D' = D + \lfloor \log_2 L \rfloor - 1$.

3. In construction of extended Merkle tree, masks $(r_l^i, r_r^i), i = 1, 2, \ldots, D'$ are used (total number of masks is $2D'$, each mask is a random string of length n). On layer i of extended Merkle tree, masks (r_l^i, r_r^i) are used to evaluate parent node. More exactly, if a_l is a left child node, and a_r is right one, then parent node is evaluated in the following way: $p = h(a_l \bigoplus r_l^i \parallel a_r \bigoplus r_r^i)$ (the same pair of masks (r_l^i, r_r^i) is used for all nodes on the layer). $2D'$ masks are published as part of public key.

As in WOTS+, masks help with constructing a reduction: now B manipulates some masks to incorporate her $\texttt{2Pre}[h]$ challenge into extended Merkle tree. Success probability of B degrades with a factor of $2^{D'}$.

Additionally, to compress a public key, in XMSS it's proposed to generate masks in a pseudorandom fashion, from some $seed'$, which becomes part of public key. Our opinion is that in this case, proof becomes not so clear: in the reduction B is allowed to choose masks, while in real scheme masks are (unpredicably) generated from a seed. This approach is actually very similar to *Random Oracle Model* (ROM) proofs.

Critiques of masking constructions. Using of masks for WOTS chains (WOTS+) and for Authentication (Merkle) trees allows to relax security requirements for underlying hash function. Critiques of this approach is based on the following theses:

1. Reduction to SPR degrades with a factor of the total number of hash evaluations in the construction (wL for WOTS+, 2^D for MSS-SPR), whereas reduction to collision resistance doesn't introduce this degradation factor.

2. These more elaborate constructions come at the expense of adding additional masks to the public key.

3. Using of masks seems like only-for-proof, "artificial" construction, without any other reasonable meaning.

4. Adding of masks brings us some sort of help in security proof (to SPR), but actually there are no known attacks which use collisions. So even breaking of CR property doesn't bring any visible help in attacking signature, at least for the current state of cryptoanalysis.

5. In case of generating masks from a seed (in order to compress public key), reduction becomes not so clear (similar to ROM proofs).

3.3 Digital signatures of MAM

3.3.1 General

As it was mentioned in Chapter 1 WOTS is used in IOTA. One-time signature usage implies permanent changing of the public key which is identified with address in block chain systems. It is not convenient to use such key infrastructure. The solution which allows to switch from one-time public key (the address) to many-time public key was suggested as long ago as 1979 year (see [149]) and is called Merkle signature scheme (MSS). This signature is detailed in Section 3.2.3. It is the approach that is used in MAM. WOTS is used as one-time signature in MSS.

3.3.2 Reconstruction

We will reconstruct the signature in the definitions of three algorithms: `KeyGen` — the algorithm for the key generation, `Sign` — the algorithm for the message signing and `Verify` — the algorithm for the signature verification. Three hash functions are used in MSS. In MAM all these hash functions (and not only they) use the ternary alphabet $\mathbf{T}$. Function $H\colon \mathbf{T}^* \to \mathbf{T}^{n_m}$ — the hash function that is used to compress the message X with an arbitrary length to the message with a length n_m. Function $h\colon \mathbf{T}^{n_w} \to \mathbf{T}^{n_w}$ — the hash function that is used in WOTS. And function $h_T\colon \mathbf{T}^{2 \cdot n_t} \to \mathbf{T}^{n_t}$ — the hash function that is used during the tree generation. The sponge hash function called Curl is used in MAM as three above hash functions. As well as any other sponge functions, Curl provides three main interfaces: `Sponge.Init`(), `Sponge.Absorb`(X) and `Sponge.Squeeze`(k). Additionally Curl provides interface `Sponge.Rate`(n) that allows to get first n trits from the internal sponge function state. `Sponge.Rate`() without parameters returns the first 243 trits. We will use above definitions in reconstruction.

3.3.2.1 Key generation

Let start reconstruction from the key generation algorithm `KeyGen`. This algorithm consists of the generation of N one-time keys and Merkle tree. The one-time private key generation is performed with key derivation function (KDF). Master-key *seed* and index of the one-time private key *index* are KDF input. The length of the private key depends on the input parameter $SL \in \{1, 2, 3\}$ — the security level of the signature. Depending on SL, the key length can have one of the following values 6561, 13122 and 19683 trits. This algorithm is described below in `MAM.WOTS.KDF`. Additional algorithm `AddAssign`(t, a) calculates the sum of a number a in trits representation and t — string of trits.

Algorithm `MAM.WOTS.KDF`

Input:

– $seed \in \mathbf{T}^{243}$ (the master key);

– $index \in \{0, 1, 2, \ldots\}$;

– $SL \in \{1, 2, 3\}$ (the security level).

Output:

– $sk \in \mathbf{T}^{6561 \cdot SL}$ (the secret key).

Steps

1. $sum \leftarrow \texttt{AddAssign}(seed, index)$.
2. `Sponge.Init`().
3. `Sponge.Absorb`(sum).
4. $subseed \leftarrow$ `Sponge.Squeeze`(243).
5. `Sponge.Init`().
6. `Sponge.Absorb`($subseed$).
7. $prekey \leftarrow$ `Sponge.Squeeze`($6561 \cdot SL$).
8. Split $prekey$ into blocks $prekey_1 \parallel prekey_2 \parallel \ldots \parallel prekey_{n_m}$, such that $|prekey_1| = \ldots = |prekey_{n_m}| = 243$ trits, $n_m = 27 \cdot SL$.
9. For $i = 1, \ldots, 27 \cdot L$ do :
   (a) `Sponge.Init`().
   (b) `Sponge.Absorb`($prekey_i$).
   (c) $sk_i \leftarrow$ `Sponge.Rate`().
10. Return $(sk_1, sk_2, \ldots, sk_{27 \cdot SL})$.

After the private key is generated the $27 \cdot SL$ blocks each of the length 243 trits are hashed (`Sponge.Init`() $\rightarrow$ `Sponge.Absorb`() $\rightarrow$ `Sponge.Squeeze`()) w times, where $w = 27$. Full algorithm is described in `MAM.WOTS.KeyGen`.

Algorithm `MAM.WOTS.KeyGen`

Input:

– $seed \in \mathbf{T}^{243}$ (the master key);

– $index \in \{0, 1, 2, \ldots\}$;

– $SL \in \{1, 2, 3\}$ (the security level).

Output:

– $sk \in \mathbf{T}^{6561 \cdot SL}$ (the secret key);

– $pk \in \mathbf{T}^{6561 \cdot SL}$ (the public key).

Steps:

1. $sk \leftarrow$ `MAM.WOTS.KDF`$(seed, index)$.
2. Split sk into blocks $sk_1 \parallel sk_2 \parallel \ldots \parallel sk_{n_m}$,
   such that $|sk_1| = \ldots = |sk_{n_m}| = 243$ trits, $n_m = 27 \cdot SL$.
3. For $j = 1, \ldots w$:
   (a) `Sponge.Init`().
   (b) `Sponge.Absorb`(sk_i).
   (c) $pk_i \leftarrow$ `Sponge.Squeeze`().
4. Return (sk, pk).

After generation of $N = 2^D$ key pairs the Merkle tree is generated. Algorithm of tree generation is called `MSS.TreeGen`. As input it gets N one-time public keys and as output it returns Merkle tree. As a result we have the following algorithm for the key generation in MAM.

Algorithm `MAM.MSS.KeyGen`

Input:

– $seed \in \mathbf{T}^{243}$ (the master key);

– $c \in \mathbb{N}$ (the number of generated one-time keys);

– $N \in \mathbb{N}$ (the number of one-time public keys);

– $SL \in \{1, 2, 3\}$ (the security level).

Output:

– $pkn \in \mathbf{T}^{(2^{D+1}-1) \times 243}$ (Merkle tree).

Steps:

1. For $i = 1, 2, \ldots N$:
   (a) $pkn_i \leftarrow$ `MAM.WOTS.KeyGen`$(seed, c + i, SL)$.
2. Return `MSS.TreeGen`(pkn).

3.3.2.2 Sign

The length of the signature depends on the security level SL and is equal $\mathbf{T}^{6561 \cdot SL}$. In the classical WOTS scheme before signature a checksum is added to the message and the message with checksum is signed. In MAM the signing message is the *tag*. Before *tag* signing the *tag* is normalised with `Search` algorithm. The *tag* construction and `Search` algorithm are detailed in Chapter 2. The algorithm `MAM.Sign` creates a signature of the message from the one-time secret key and the security level. *Auth* that helps to verify the signature is created by the index of one-time private key on the upper level. Additional algorithm `TryToInt` returns integer representation of one tryte.

Algorithm `MAM.Sign`

Input:

- $SL \in \{1, 2, 3\}$ (the security level);
- $sk \in \mathbf{T}^{6561 \cdot SL}$ (the secret key);
- $M \in \mathbf{T}^{81 \cdot SL}$ (the signing message).

Output:

- $\sigma \in \mathbf{T}^{6561 \cdot SL}$ (the signature of the message).

Steps:

1. $n_m \leftarrow 27 \cdot SL$.
2. Split sk into blocks $sk_1 \parallel sk_2 \parallel \ldots \parallel sk_{n_m}$.
3. Split M into blocks $M_1 \parallel M_2 \parallel \ldots \parallel M_{n_m}$.
4. For $i = 1, 2, \ldots n_m$:
   (a) for $j = 1, \ldots$ `TryToInt`(M_i):
      i. `Sponge.Init`().
      ii. `Sponge.Absorb`(sk_i).
      iii. $\sigma_i \leftarrow$ `Sponge.Squeeze`().
5. Return σ.

3.3.2.3 Verify

The verification of signature in the classical MSS consists of two steps: WOTS verification and Merkle tree verification. In case of MAM WOTS verification is not used because one-time public key is not the part of the signature. One-time public key is calculated from the signed message and then this value is used to create root from it and *Auth*. If calculated $root'$ is equal to original *root* then the signature is valid. The algorithm `MAM.Verify` verifies the signature by the Merkle tree root and the authentication path and is described below. Additional algorithm `CreateRoot` creates Merkle tree root from the authentication path *Auth* and the index of

the one-time private key *index*. Additional algorithm `GetSL` returns the secret level from the message M.

Algorithm MAM.Verify

Input:

- $M \in \mathbf{T}^{243}$ (the signing message);
- $pk \in \mathbf{T}^{243}$ (the root of Merkle tree);
- $\sigma \in \mathbf{T}^{6561 \cdot SL}$ (the signature);
- $Auth \in \mathbf{T}^{243 \cdot (D-1)}$;
- $index \in \{0, 1, 2, ..\}$ (the index of WOTS instance in the tree).

Output:

- 1 (valid) or 0 (not valid).

Steps:

1. $SL \leftarrow \texttt{GetSL}(M)$.
2. $n_m \leftarrow 27 \cdot SL$.
3. Split σ into blocks $\sigma_1 \parallel \sigma_2 \parallel \ldots \parallel \sigma_{n_m}$.
4. Split M into blocks $M_1 \parallel M_2 \parallel \ldots \parallel M_{n_m}$.
5. For $i = 1, 2, \ldots n_m$:
   (a) for $j = 1, \ldots \texttt{TryToInt}(w - \texttt{TryToInt}(M_i))$:
      i. `Sponge.Init`().
      ii. `Sponge.Absorb`(σ_i).
      iii. $otpk_i \leftarrow$ `Sponge.Squeeze`().
6. `Sponge.Init`().
7. `Sponge.Absorb`($otpk$).
8. `Sponge.Absorb`(`CreateRoot`($Auth, index$)).
9. $pk' \leftarrow$ `Sponge.Squeeze`().
10. Match pk' to pk: if $pk' = pk$, then return 1, otherwise — return 0.

3.3.3 Analysis

Let us determinate two directions of analysis. The first direction is called Secure balance. This problem is that used hash functions H and h are not well-balanced and one of sizes may be decreased standing the same strength of WOTS signature. The second problem is so-called normalisation algorithm. This algorithm is used in MAM instead of classic WOTS checksum and changes the singing message in a tricky way. Let see at this problems deeply.

3.3.3.1 Secure balance

The easiest problem which stands for the adversary Mallory who attacks MSS is to win EU-CMA1 game. The description of this game is introduced in Section 3.1. Suppose that Mallory wants to forge a signature of a message X such that $H(X)$ is not previously processed in WOTS. Then, due to the structure of WOTS, Mallory has to solve at least one of the following problems: $\texttt{Inv}[h]$, $\texttt{2Pre}[h]$ or $\texttt{Coll}[H]$. These problems also are introduced in Chapter 1. Solving these problems we should keep in mind quantum algorithms because Quantum-Proof-Crypto, that is, cryptography which is resistance against quantum computers, is one of the cornerstones of MAM.

Using the time evaluations for solving $\texttt{Inv}[h]$, $\texttt{2Pre}[h]$ and $\texttt{Coll}[H]$ from Chapter 1, we have the following values for the strength of WOTS in MAM: $\min(O(w^{n_w}),O(w^{n_m/2}))$ — Pre-Quantum epoch, $\min(O(w^{n_w/2}),O(w^{n_m/3}))$ — Post-Quantum epoch. As in MAM there are three levels of the signature with different sets of parameters so let us evaluate the strength of WOTS for all these sets. The evaluations are presented in Table 3.1.

Table 3.1: The strength of WOTS in MAM

Parameters	Pre-Quantum	Post-Quantum
$n_w = 81, n_m = 27$	2^{64}	2^{42}
$n_w = 81, n_m = 54$	2^{128}	2^{85}
$n_w = 81, n_m = 81$	2^{192}	2^{128}

To begin with, the first level ($n_w = 81, n_m = 27$) of the signature is weak even nowadays. For comparison, every 10 minutes all Bitcoin miners are producing approximately 2^{68} operations. As for the second and the third levels, they have one common problem: their parameters are unbalanced because n_w is very big. The changing of n_w to 54 does not change the strength of WOTS. Finally, the second level of MAM signature is weak in Post-Quantum epoch. As quantum-proof signature algorithm is used in MAM it would be correct to stay strong in Post-Quantum epoch.

3.3.3.2 Normalisation algorithm

To achieve Property 1 checksum is used in classic WOTS. Instead of this, a tricky algorithm with *nonce* — the word of 27 trytes — is used in MAM. This algorithm is described in Section 2.5.2 and called normalisation (in reconstruction it names `Search`). The length of the message n_m depends on the security level of the signature and can equal 27, 54 or 81 trytes. After the normalisation the sum of all trytes of the message equals 0: $\sum_{i=0}^{n_m-1} M_i = 0$. If the

length of the message is 54 trytes then the sum of the first 27 trytes must not be equal 0. Accordingly, if the length of the message is 81 the sum of the first 27 and 54 trytes must not be equal 0. Such a construction of the normalised message leads to a decrease in the number of normalisation function images. We say "the number of normalisation function images" because the normalisation function is in fact a hash function. Let us evaluate this number.

The values for the number of messages where the sum of all trytes equals zero is very large. Thus it is difficult to evaluate this value with standard discrete formulas without large amount of machines calculations. So for this purpose we use Gnedenko's local limit theorem [98]. Let write this theorem in the following form: Let $\xi_1, \xi_2, \ldots, \xi_n$ are independent and identically distributed random variables with lattice distribution and mean $E\{\xi_i\} = \mu$, variance $D\{\xi_i\} = \sigma^2 < \infty$. Then

$$\mathbf{P}\{S_n = k\} = \frac{1}{\sqrt{2\pi n}\sigma} \cdot e^{\frac{-(k-\mu \cdot n)^2}{2 \cdot n \cdot \sigma}} + o(\frac{1}{\sqrt{n}})$$

where $S_n = \xi_1 + \xi_2 + \ldots + \xi_n$, $\mathbf{P}\{S_n = k\}$ — the probability that the sum of n random variables is equal k.

In our case $\xi_i \in \{-13, -12, -11, \ldots, 13\}$, $i = 1, 2, \ldots, n$ are random variables with lattice distribution where span equals 1, $n = n_m$, $k = 0$, $\mu = 0$, $\sigma^2 = \sum_{i=1}^{13} 2 \cdot i^2/27 \approx 60.67$. The second factor in above formula is always equals 1, because k and μ are equal 0. Then we have $\mathbf{P}\{S_n = 0\} = 1/\sqrt{2\pi n}\sigma$. Since $\mathbf{P}\{S_n = 0\} = N_0/N_{all}$, where N_0 — the number of normalisation images, N_{all} — the number of all messages with the length n_m, is another evaluation for $\mathbf{P}\{S_n = 0\}$, then we have $N_0 = N_{all} \cdot \mathbf{P}\{S_n = 0\}$. The number of all messages N_{all} is equal 27^{n_m}. The $\mathbf{P}\{S_n = 0\}$ may be perceived as the probability of successful *nonce* generation.

All evaluations of the probabilities are gathered in Table 3.2. The number of messages where the sum of all trytes equals zero is very large so there is no sense to include this values into the table.

Table 3.2: Probabilities of success for normalisation algorithm

n_m	The probability of success	The number of iterations to success
27	0.0099	102
54	0.0070	144
81	0.0057	176

One of the main problem that we solve in our team was choosing of way to achieve Property 1. As we discussed above there are two ways: normalisation and checksum. Checksum is a classic solution that is used in all WOTS schemes. Normalisation is MAM innovational algorithm. Each of these algorithms has some advantages and disadvantages. Let us present them in Table 3.3.

Based on this table we decided to use checksum.

Table 3.3: Normalisation and checksum comparison

Algorithm	Pro	Contra
Normalisation	No strength loosing. Proof of work.	Additional encryption. Extra space in message. Indefinite search time. Difficult implementation.
Checksum	Easy implementation.	Strength loosing. Extra WOTS chains.

3.4 Digital signatures of MAM2

3.4.1 Concept

In this chapter we present a digital signature algorithm for MAM2. Generally, it's a MSS[WOTS] scheme (see 3.2.3), with the following details:

1. Ternary alphabet is used: $\mathbf{A} = \mathbf{T}$.
2. For WOTS, parameter $w = 27$.
3. Size of message digest $n_m = 234$, or 78 trytes.
4. Hash length in Winternitz chains $n_w = 162$, or 54 trytes.
5. Plain WOTS chains are used (no masks like WOTS+).
6. In WOTS, checksums of length 9 are used (3 trytes), making total number of chains $L = 81$.
7. Plain Merkle tree is used (no masks like SPR-MSS).
8. Height of Merkle tree is arbitrary up to 20: $0 \leqslant D \leqslant 20$. The choice of D depends on the computational possibilities of the devices and the number of required keys. $D = 0$ means that scheme degrades to just one-time WOTS.
9. Hash length in Merkle tree $n_t = 243$, or 81 trytes.
10. To build a leaf from WOTS public key pk, hash function h_T (the same as for Merkle tree) of concatenation of public key components is evaluated: $leaf = h_T(pk_1 \parallel pk_2 \parallel \ldots \parallel pk_L)$.
11. Secret key is generated in a pseudorandom fashion from a single value *seed*.
12. Signature of message X is a tetrad $(D, index, \sigma^{WOTS_i}, Auth)$. The encoding is described later.

Analysis of decisions is given in 3.4.2.

Encoding of signature value. Signature of message X is a tetrad $(D, index, \sigma^{WOTS_i}, Auth)$. It's represented as an array of trytes of some length, depending on chosen D:

1. The first 4 trits encode tree height D (it's enough to encode heights up to 20).
2. The next 14 trits encode MSS index $index$ (it's enough to encode up to 2^{20} indices).
3. The next 13122 trits (i.e 4374 trytes) represents WOTS signature σ^{WOTS_i}: we have 81 chains, and each chain implies value of 54 trytes.
4. The remaining trytes encode Merkle authentication path $Auth$. For tree height D we need $243D$ trits (i.e. $81D$ trytes): authentication path consists of D nodes, each requires 81 trytes.

Comparison to MAM digital signature. One can see MAM2 digital signature algorithm as an upgrading of the existing solution (MAM). First of all, balancing of hash sizes has been made: for H we use 78 trytes, for h_T — 81 trytes, while for h — 54 trytes. These sizes provide the security level of the signature about 2^{192} in Pre-Quantum epoch and 2^{128} in Post-Quantum Epoch. The problem of not-balanced sizes in MAM is described in 3.3.3.

Then, checksum is used instead of normalisation algorithm in WOTS of MAM2.

3.4.2 Analysis of decisions

In this section decisions of MAM2 digital signature design are explained.
Our efforts were driven by the following aims:

1. We need a provably secure digital signature scheme.
2. The scheme should be *balanced*, which means that security levels of all cryptographic primitives and subsystems should be (almost) equal.
3. The aimed security level is 2^{128}, which is a popular choice for cryptographic systems nowadays.
4. The scheme needs to be secure in Post-Quantum Epoch.
5. The scheme should be effective. The most important properties to minimize are signature size and verification time. The former is because of pretty limited (or at least expensive) bandwidth of distributed ledger technology such as IOTA, while the latter is due to a focusing of IOTA/MAM to IoT low-energy devices.

Message digest size. For message digest size, we use 78 trytes ($n_m = 234$), which equals to 371 bits. Since H requires to be CR (see 3.2.5), and there's a generic attack against CR in Post-Quantum Epoch which takes $O(N^{1/3})$ time (see 1.3.4), the post-quantum security level is about 2^{124}. While it's a bit less than aimed level of 2^{128}, we have convenient number of chains in WOTS (81, if take into account additional 3 trytes of checksum).

Winternitz chains. For Winternitz chains, original WOTS construction is used, based on simple hash chain (e.g. without masks like in WOTS+). In solutions like WOTS+, more complicated chain constructions are employed to get rid of CR requirement for hash function in security proof (see 3.2.5). We provide strong arguments that even plain WOTS doesn't require CR hash functions, and security of plain WOTS is almost the same as e.g. WOTS+, in 3.5. So, this result allows us to make h compact: only 54 trytes are required ($n_w = 54 \cdot 3 = 162$), which corresponds to 2^{128} security level in Post-Quantum Epoch (Grover's algorithm of $O(N^{1/2})$ time is taken into account, see 1.3.4).

Merkle tree hash size. For Merkle tree in MSS construction, we use plain hash-tree construction, without masks like SPR-MSS (see 3.2.5). To keep our scheme provably secure, we need CR hash function for building tree: although there's a guess that CR is not required, we was not able to construct similar arguments as we did for Winternitz chains. Perhaps, it's a matter of time, and such arguments will appear. But now, to stay in provable security, we use hash function with size of 81 trytes ($n_t = 81 \cdot 3 = 243$), which corresponds to 2^{128} security level in Post-Quantum Epoch (collision-finding quantum algorithm of $O(N^{1/3})$ time is taken into account, see 1.3.4). Although we was not able to compress Merkle tree nodes, it's not so a big flaw: anyway, in whole signature value, WOTS part prevails.

Checksum in WOTS. Checksum is not used in MAM. The normalisation algorithm is used instead of it. The analysis of normalisation is given in 3.3.3. Normalisation is potentially interesting approach that solves the same problem (Property 1) as the checksum but in another way. But we decided to use checksum because it is a simple and stable approach, while additional security analysis for normalization is needed.

Algorithm for finding checksum is described in 3.2.2. For $n_m = 234$ (in other words, 234**T** message digest) we need 9**T** checksum, which in total requires 81 Winternitz chains.

3.4.3 Security of the scheme

The scheme is provably secure in EU-CMA model (see 3.1). A proof that MSS[WOTS] is secure if CR hash functions are used is a well known result (e.g. see [26]). We impoved the result, providing strong arguments that CR is not required for WOTS part of the scheme. For details, see 3.5.

Exact security level is defined by the weakest unit of the scheme. In our case, this is an attack against CR property of H. We estimate 2^{187} security level in Pre-Quantum Epoch (taking into account so-called *birthday-paradox attack*), and 2^{124} in Post-Quantum Epoch (see 1.3.4).

It worth to note, that these estimates could be not exact, taking into account so-called *multi-target attacks*, introduced in [120]. However security level will drop just insignificantly (taking into account not large Merkle tree height), so we can ignore it.

3.4.4 The Merkle tree depth

The only parameter (chosen by a user) in the scheme is the Merkle tree depth D. In this section, recommendations for choosing the value of D are provided.

The depth of the Merkle tree D defines the number of one-time keys in it. Choosing the depth you should take into account tree generation time. The more the depth is, the more

time is needed. The tree generation consists of $2^D - 1$ hash evaluations and 2^D WOTS key generations. WOTS key generation consists of $n_w \cdot (w-1)$ hash evaluations. To simplify the calculations we suppose that the same hash function is used in WOTS instances and for Merkle tree generation. We will consider Keccak is this hash function. Its main characteristics are taken from [146]. They include cycles per byte (cpb) values for different processors. Since MAM2 is seen as a solution for Internet of Things (IoT) we consider two types of devices: IoT devices and PCs. IoT device is a 4 x 1704MHz ARM processor of 2012 year. PC is a 2 x 3800MHz AMD processor of 2012 year. As these processors are released in 2012 we suppose that they are the devices of average users. We use cycles per byte (cpb) values for 64 bytes input. For IoT device it is 86.25 cpb. For PC – 26.08 cpb.

We have the following formula for estimation of WOTS key generation time:

$$T_{key} = \frac{N_{cycles}}{freq} = \frac{n_m \cdot (w-1) \cdot n_w \cdot cpb}{freq},$$

where $n_m = 81$ trytes, $n_w = 54$ trytes ≈ 33 bytes, $w = 27$, cpb – cycles per byte for processor, $freq$ – the frequency of a processor. As a result, we have tree generation time:

$$T_{tree} = T_{key} \cdot 2^D + \frac{(2^D - 1) \cdot 2 \cdot n_t \cdot cpb}{freq} \approx T_{key} \cdot 2^D.$$

Results of calculations are gathered in Table 3.4.

Table 3.4: Tree generation time comparison

Depth	Number of keys	Generation time (IoT device), s	Generation time (PC), s
20	1000000	9567	4290
19	500000	4783	2145
18	250000	2392	1073
17	125000	1195	536
16	62500	597	268
15	31250	299	134
14	15625	150	67

Tree generation time is pretty large for $D > 16$, but maybe it's still practical for some users.

It's worth to note that all estimations are approximate. In practice, a processor can give only a part of its resources to the tree generation algorithm, so it can take more time.

3.5 Security of WOTS

3.5.1 Background

Let us use notations and definitions introduced in Chapter 1. Let WOTS$[H, h]$ be the WOTS scheme instantiated with hash functions $H\colon \mathbf{T}^* \to \{0, 1, \ldots, w-1\}^n$ and $h\colon S \to S$.

When justifying the strength of WOTS, it is necessary to show that a hypothetical algorithm A which solves the problem $P_1 = \texttt{EU-CMA}$ with respect to WOTS$[H, h]$ can be transformed

with insignificant expenses into an algorithm B which solves some difficult problem P_2 regarding H and h. In such cases we say that P_2 is *reduced* to P_1: $P_2 \leqslant P_1$. The phrase "*with insignificant expenses*" means that the running time of B is insignificantly larger than the running time of A and the probability of success is negligible smaller. Formal definitions regarding reduction in complexity theory can be found, for example, in [100].

Note that the notation $P_2 \leqslant P_1$ for reduction is quite suitable: indeed, P_1 is not *weaker* than P_2. Therefore, if P_2 is difficult (and this is so if cryptographically strong H and h are used), then P_1 is also difficult. Mathematically speaking, reduction sets a partial preorder on a set of computational problems and this preorder can be used to justify the complexity of one problem knowing the complexity of other.

As usual in cryptography, an algorithm that solves a cryptanalytic problem is called an *adversary*. An adversary A gets a public key $pk = (pk_1, \ldots, pk_n)$ build upon a private key $sk = (sk_1, \ldots, sk_n)$ which has been chosen at random from S^n. The adversary can choose an arbitrary message X and receive the corresponding signature $s = (s_1, \ldots, s_n)$ in which

$$s_i = h^{m_i}(sk_i), \quad i = 1, 2, \ldots, n.$$

Here m_i is the ith symbol of $H(X)$. After receiving s the adversary must find a message $X' \neq X$ and its signature $s' = (s'_1, \ldots, s'_n)$ which will be accepted:

$$h^{w-1-m'_i}(s'_i) = pk_i, \quad i = 1, 2, \ldots, n.$$

Here m'_i is the ith symbol of $H(X')$.

The adversary really has to *forge* a signature, that is, to find it without knowing the private key sk.

Two cases are possible:

1. $H(X) = H(X')$ and, therefore, the adversary has solved $\texttt{Coll}[H]$.

2. $H(X) \neq H(X')$. Due to Property 1 of H the inequality $m'_i < m_i$ holds at least for one $i \in \{1, 2, \ldots, n\}$. It means that the adversary has found on her own (without quering A) the m'_ith preimage of pk_i:
$$h^{m'_i}(s'_i) = pk_i.$$
In particular, the adversary has solved the problem $\texttt{Inv}[h](pk_i)$, that is, has found a (first) preimage of pk_i:
$$h(pk''_i) = pk_i,$$
where $pk''_i = h^{m'_i - 1}(s'_i)$.

It seems that the arguments above are sufficient to justify the security of WOTS because to forge a signature, an adversary needs to solve either a hard problem $\texttt{Coll}[H]$ or the hard problem $\texttt{Inv}[h]$. Unfortunately, the arguments are not fully satisfactory. The problematic point is that A solves $\texttt{Inv}[h]$ with *some* input pk_i. The arguments would be final if we managed to use A to solve $\texttt{Inv}[h]$ with an *arbitrary* (chosen) input y.

Let this input receive another adversary B. He needs to solve $\texttt{Inv}[h](y)$ with a permission to access A. Let $y = h(x)$, where x is chosen at random from S.

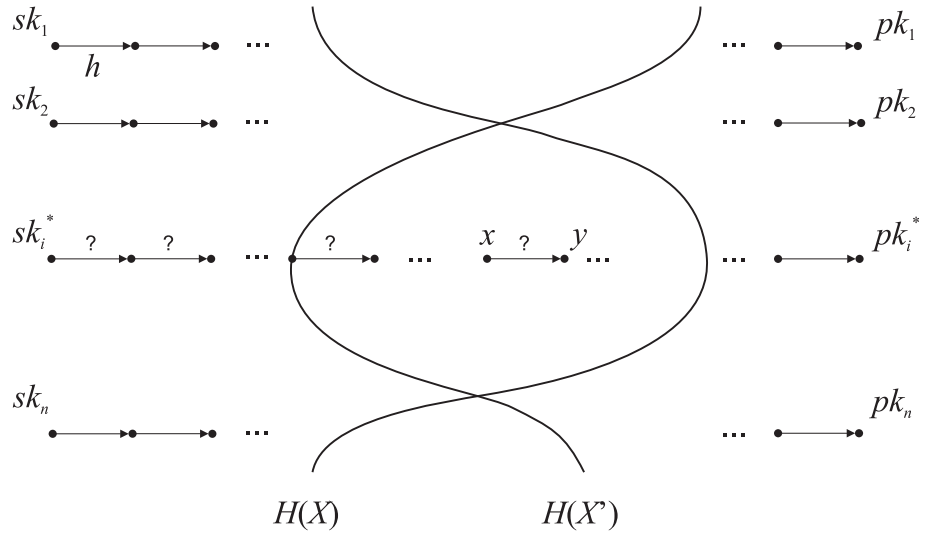

Figure 3.1: Embedding of y

The adversary B acts as follows (see Figure 3.1):

1. Generates a pair (sk, pk) in usual manner and then embeds y somewhere "between" sk and pk. For example, B chooses $i \in_R \{1, \ldots, n\}$, $t \in_R \{1, \ldots, w-1\}$ and changes $pk_i = h^{w-1}(sk_i)$ by $pk_i^* = h^{w-1-t}(y)$. In other words, B represents y as $h^t(sk_i^*)$ for some unknown (and maybe even non-existent) partial private key sk_i^*.

2. Starts the algorithm A passing the public key pk to it.

3. Waits for a signature query X from A. Returns $\perp$ if the query cannot be processed. For example, y has been embedded as described above and $m_i < t$.

4. Waits for a response from A. This response is either a pair (X', s') or the error code $\perp$. If A returns $\perp$, then B finishes repeating this answer.

5. Analyzes the pair (X', s') and determines from it a solution of $\texttt{Inv}[h](y)$. Returns $\perp$ if a solution cannot be determined. For example, if y was embedded as described above, then a solution will be found if $m_i' < t$ and $h^{t-m_i'}(s_i') = y$. Indeed, in this case $x = h^{t-m_i'-1}(s_i')$ is the required solution.

Using reasonable embeddings of y (including the embedding described above), we can expect that the probability of success of B drops insignificantly with respect to the probability of success of A and that we really get the reduction of the composite problem $\texttt{Coll}[H] \vee \texttt{Inv}[h]$ to $\texttt{EU-CMA}[\mathrm{WOTS}[H, h]]$.

However, there remain two essential difficulties. First, an embedding of y should not change the probability distribution of pk. Otherwise B can not guarantee the preservation of probability of success A after the embedding. In particular, the described embedding generally changes the distribution: pk_i^* is the $(w-t)$th image (with respect to h) of $x \in_R S$, while pk_i is the $(w-1)$th image of $sk_i \in_R S$. Second, at the last step B has no guarantees of a benefit of A's response. For example, for $m_j' < m_j$ an adversary A can never return s_j' from which s_j can be obtained by iterating h (although pk_j is necessarily obtained). In other words, the equality $h^{m_j - m_j'}(s_j') = s_j$ and, in particular, the equality $h^{t-m_i'}(s_i') = y$ can never be fulfilled.

It is necessary to make considerable efforts to overcome these difficulties. For example, in the well-known scheme WOTS+ [119] its inventor:

- uses the even more complicated composite problem $\texttt{Coll}[H] \vee \texttt{Inv}[h] \vee \texttt{2Pre}[h]$;
- randomizes preimages of h during iterations (using masks);
- brings the assumption that images of h are indistinguishable from instantiations of independent random variables with a uniform distribution over S;
- changes a specific hash function h to a random representative h_k of a family of hash functions.

In fact, we are talking about a composition of *one-time* independent random functions for generation of *one-time* signatures. In our opinion, WOTS+ is a somewhat artificial construction.

The next popular construction — WOTS-PRF [53] — seems more natural. Unfortunately, its proof of security turned out to be incorrect — the paper [134] reports about an error.

Further we propose an alternative plan of reduction. The main idea is to use other hard problems regarding h. We don't change WOTS at all.

3.5.2 The Hook problem

Let us introduce the following

Problem Hook

Parameters: h — a hash function $S \to S$, r — a small positive integer.

Input: $x \in S$. Further we assume that x is chosen at random from S.

Output: $x' \in S$ such that

$$x' \neq h^{t-1}(x), \quad h(x') = h^t(x),$$

for some $t \in \{1, 2, \ldots, r\}$ – a *level* of the solution.

In other words, in the problem $\texttt{Hook}[h, w](x)$ we need to find any vertex x' adjacent to the h-path of x of length r and not lying on this path. The chosen name "hook" is motivated by the corresponding graphical configuration (see Figure 3.2).

Figure 3.2: A hook

It is clear that $\texttt{Hook}[h, 1]$ coincides with $\texttt{2Pre}[h]$. It is also clear that

$$\texttt{Hook}[h, r] \leqslant \texttt{2Pre}[h].$$

Indeed, to solve $\texttt{Hook}[h, r](x)$ we can choose $t \in \{1, 2, \ldots, r\}$, then run an algorithm which solves $\texttt{2Pre}[h](h^t(x))$ and returns its answer x' as our own. If the algorithm doesn't answer, then we go to the next t and so on.

Additionally,

$$\texttt{Coll}[h] \leqslant \texttt{Hook}[h, r].$$

Indeed, if x' is a solution of $\texttt{Hook}[h, r](x)$, then one of the pairs $(x', h^{t-1}(x))$, $t = 1, 2, \ldots, r$, gives a collision:

$$h(x') = h^t(x) = h(h^{t-1}(x)).$$

Having restrictions

$$\texttt{Coll}[h] \leqslant \texttt{Hook}[h, r] \leqslant \texttt{2Pre}[h],$$

we should understand to which problem `Hook` is "closer": to the "weak" `Coll` or to the "strong" `2Pre` (see Table 1.3).

To understand the difference between $\texttt{Coll}[h]$ and $\texttt{2Pre}[h]$, let us introduce an equivalence relation over S: $x \sim x'$ if $h(x) = h(x')$. The relation $\sim$ splits S into equivalence classes. Let C be a system of such classes: $S = \cup_{c \in C} c$. Let $c(x)$ be the class to which x belongs.

The problem $\texttt{Coll}[h]$ can be formulated as follows: Find distinct x and x' that belong to the same class c. The problem $\texttt{2Pre}[h]$ sounds different: Find $x' \neq x$ in the class $c(x)$. The difference between the problems is essential. In the first case the class c can be chosen, in the second case the class is given. The possibility to choose just determines the comparative simplicity $\texttt{Coll}[h]$ relative to $\texttt{2Pre}[h]$.

A slightly strengthened version of $\texttt{Hook}[h, r]$ can be formulated in the same way as $\texttt{2Pre}[h]$ with only difference in definition of equivalence classes: Find $x' \neq x$ from the equivalence class $c_r(x)$. Here c_r is defined by a new relation $\sim_r$: $x \sim_r x'$ if $h^r(x) = h^r(x')$. As in $\texttt{2Pre}[h]$, the searching of x' is performed in the *given* class $c_r(x)$, the only difference is that this class is generally wider than the previous class $c(x) = c_1(x)$.

Let $N = |S|$ and let a random instance of $\texttt{Hook}[h, r]$ have M solutions on average. Then $\texttt{Hook}[h, r]$ can be solved in average time $O(\sqrt{N/M})$ using Grover's quantum algorithm (see 1.3.4). Further we show that for a strong (and, consequently, similar to random) hash function h and for a small r the number of solutions $M = M(h, r)$ is small, and therefore the time to solve $\texttt{Hook}[h, r]$ is presumably slightly less than the time to solve $\texttt{2Pre}[h]$.

3.5.3 Compression coefficients

Let S_t be a set of images achievable after t iterations of h:

$$S_0 = S, \quad S_t = \{h(x) \colon x \in S_{t-1}\}, \quad t = 1, 2, \ldots.$$

It is well known (see, for example, [89, Theorem 2]) that if h is a random (ideal) hash function and t is small, then $|S_t|$ is close to

$$|S|(1 - \tau_t),$$

where τ_t is recursively defined as: $e^{\tau_{t-1} - 1}$, $\tau_0 = 0$.

Call the quantities $k_t = |S_t|/|S_{t-1}|$, $t = 1, 2, \ldots$, *compression coefficients*. The first 26 coefficients of an ideal function h are given in Table 3.5.

Table 3.5: Compression coefficients of an ideal h

t	$1-\tau_t$	k_t	t	$1-\tau_t$	k_t
1	0.6321	0.6321	14	0.1195	0.9389
2	0.4685	0.7412	15	0.1126	0.9425
3	0.3740	0.7984	16	0.1065	0.9457
4	0.3120	0.8342	17	0.1010	0.9485
5	0.2680	0.8590	18	0.0961	0.9511
6	0.2351	0.8771	19	0.0916	0.9534
7	0.2095	0.8911	20	0.0875	0.9555
8	0.1890	0.9021	21	0.0838	0.9574
9	0.1722	0.9111	22	0.0804	0.9592
10	0.1582	0.9186	23	0.0772	0.9608
11	0.1463	0.9248	24	0.0743	0.9623
12	0.1361	0.9302	25	0.0716	0.9637
13	0.1272	0.9349	26	0.0691	0.9650

As we see, k_t grow monotonically approaching 1 but still remain sufficiently below 1 for small t. This means that the probability of finding a solution of $\texttt{Hook}[h,r]$ at each level $t = 1,2,\ldots,r$ is quite high.

The average number M of solutions of $\texttt{Hook}[h,r](x)$ can be upper bounded by the average cardinality (over random x) of $c_r(x)$. If h is ideal, then this average cardinality is close to $1/(1-\tau_r)$. For example, for $r = 26$ the number M is upper bounded by $1/0.0691 \approx 14.5$. The number of solutions of $\texttt{Hook}[h,r]$ is really small on average.

3.5.4 Reduction

In $\texttt{Inv}[h]$ the input y (an element which has to be inverted) is usually chosen as $y = h(x)$, where $x \in_R S$. In the modified problem $\texttt{Inv}[h,r]$ the input is a pair (y,t) in which $t \in_R \{1,2,\ldots,r\}$ and $y = h^t(x)$. In both problems an adversary has to find $x' \in S$ such that $h(x') = y$.

The problems differ only in the way that the input y is generated. It seems that for a strong (similar to the random) function h the problems have approximately the same complexity. It is also clear that $\texttt{Inv}[h,1] = \texttt{Inv}[h]$.

Theorem 3.1. *Let $WOTS[H,h]$ be the WOTS scheme instantiated with hash fuctions $H\colon \boldsymbol{T}^* \to \{0,1,\ldots,w-1\}^n$ and $h\colon S \to S$. If there exists an algorithm A which solves the problem $\texttt{EU-CMA}[WOTS[H,h]]$ in time T with probablility α, then there exists an algorithm B which solves one of the problems $\texttt{Coll}[H]$, $\texttt{Hook}[h,w-1]$, $\texttt{Inv}[h,w-1]$ in time $T + O(nw)$ with probability*

$$\beta \geqslant \frac{\alpha}{2n(w-1)}.$$

Proof. The algorithm B receives the problems $\texttt{Hook}[h,w-1](x)$ and $\texttt{Inv}[h,w-1](y,t)$. In the first problem $x \in_R S$. In the second problem $t \in_R \{1,2,\ldots,w-1\}$, $y = h^t(x)$ where $x \in_R S$ is unknown.

The algorithm B randomly chooses a number $d \in_R \{1,2\}$ of the problem to be solved. With any choice B generates keys (sk, pk) of WOTS$[H,h]$, selects $i \in_R \{0,1,\ldots,n-1\}$ and embeds the chosen problem in the line $sk_i, h(sk_i), \ldots, pk_i = h^{w-1}(sk_i)$. The algorithm B takes time $O(nw)$ to generate (sk, pk), to embed the chosen problem, to interact with A and to process its response.

If $d = 1$ then B changes sk_i to x and pk_i to $h^{w-1}(x)$. If $d = 2$ then the chain $h^t(sk_i), h^{t+1}(sk_i), \ldots, pk_i$ is changed to $y, h(y), \ldots, h^{w-1-t}(y)$. Let us implicitly replace the chain $sk_i, h(sk_i), \ldots, h^{t-1}(sk_i)$ by the chain $x, h(x), \ldots, h^{t-1}(x)$, where x (unknown) determines the input y of $\mathtt{Inv}[h, w-1](y,t)$.

For any d the B's embedding is statistically indistinguishable.

The algorithm B passes A the public key pk with the embedded problem and waits for the query X such that $H(X) = m_1 m_2 \ldots m_n$. At $d = 1$, B surely responds to the request: it knows all the private keys including x which is built-in as sk_i. At $d = 2$, B can answer the query only if $t \leqslant m_i$. We estimate the probability of this inequality being fulfilled a bit later.

After B has responded to the A's query it expects (X', s') from A. Receiving a forgery (with probability α) B analyzes it.

First, the equality $H(X) = H(X')$ is verified. If it holds, then B returns a pair (X, X') as a solution of $\mathtt{Coll}[H]$. In what follows we assume that $H(X) \neq H(X')$, that is, $m_1 m_2 \ldots m_n \neq m'_1 m'_2 \ldots m'_n$.

If $d = 2$, then B verifies if the inequality $m'_i < t$, additional to $t \leqslant m_i$, holds. Probability of both inequalities:

$$\mathsf{P}\{m'_i < t \leqslant m_i\} = \mathsf{P}\{m'_i < m_i\}\mathsf{P}\{t \in \{m'_i + 1, \ldots, m_i\} \mid m'_i < m_i\} \geqslant \frac{1}{n(w-1)}.$$

Here we use the estimate

$$\mathsf{P}\{m'_i < m_i\} \geqslant \frac{1}{n}.$$

Indeed, by Property 1 there exists $j \in \{1, 2, \ldots, n\}$ such that $m'_j < m_j$. A random index i will be one of such j with probability at least $1/n$.

Then B goes to the analysis of the coordinate s'_i of the forged signature s'. Since s' is correct,

$$h^{w-1-m'_i}(s'_i) = h^{w-1}(x),$$

where x is either known ($d = 1$) or unknown ($d = 2$).

If s'_i doesn't belong to the path $x, h(x), \ldots, h^{w-1}(x)$, and $d = 1$, then the problem $\mathtt{Hook}[h, w-1](x)$ is solved. Indeed, there exists $t \in \{m'_i, \ldots, w-1\}$ such that for $x' = h^{t-m'_i}(s_i)$ it holds: $x' \neq h^{t-1}(x)$, $h(x') = h^t(x)$. The success probability

$$\alpha \cdot \mathsf{P}\{d = 1\} = \frac{\alpha}{2}.$$

If s'_i lies on the path and $d = 2$, then for $x' = h^{t-m'_i-1}(s_i)$ the equality $h(x') = y$ holds, that is, the problem $\mathtt{Inv}[h, w-1](y,t)$ is solved. The success probability

$$\alpha \cdot \mathsf{P}\{d = 2\} \cdot \mathsf{P}\{m'_i < t \leqslant m_i \mid d = 2\} \geqslant \frac{\alpha}{2n(w-1)}.$$

□

Further we show that the problem $\texttt{Hook}[h, r]$ in the theorem can be replaced by the problem $\texttt{2Pre}[h, r]$. The input of the last problem is a pair (z, t) in which $t \in_R \{0, 1, \ldots, r-1\}$ and $z = h^t(x)$ for $x \in_R S$. The answer: $z' \neq z$ such that $h^{r-t}(z') = h^{r-t}(z)$.

The solution $z' = \texttt{2Pre}[h, r](z, t)$ is also a solution of $\texttt{2Pre}[h^r]$:

$$h^{r-t}(z') = h^{r-t}(z) \quad \Rightarrow \quad h^r(z') = h^r(z).$$

In $\texttt{2Pre}[h^r]$, the input z is usually generated as $z = h^r(x)$, where $x \in_R S$. As we see, the difference is only in the number of iterations which are applied to a random x.

It seems that for a strong h the problems $\texttt{2Pre}[h, r]$ and $\texttt{2Pre}[h^r]$ are of the same complexity and the problems $\texttt{2Pre}[h^r]$ and $\texttt{2Pre}[h]$ are also approximately equally difficult.

Theorem 3.2. *Let $WOTS[H, h]$ be the WOTS scheme instantiated with hash fuctions $H\colon \boldsymbol{T}^* \to \{0, 1, \ldots, w-1\}^n$ and $h\colon S \to S$. If there exists an algorithm A which solves the problem $\texttt{EU-CMA}[WOTS[H, h]]$ in time T with probablility α, then there exists an algorithm B which solves one of the problems $\texttt{Coll}[H]$, $\texttt{2Pre}[h, w-1]$, $\texttt{Inv}[h, w-1]$ in time $T + O(nw)$ with probability*

$$\beta \geqslant \frac{\alpha}{2n(w-1)}.$$

Proof. The algorithm B acts approximately in the same way as in the previous proof. The only difference is that for $d = 1$ the problem $\texttt{2Pre}[h, w-1](z, t)$ not $\texttt{Hook}[h, w-1](z, t)$ is embedded: $y = h(z) = h^t(x)$ is processed exactly in the same way as for $d = 2$.

The query and response of A for $d = 1$ are processed in the same way as for $d = 2$. The condition $m'_i < t \leqslant m_i$ is fulfilled with probability at least $1/(n(w-1))$.

Depending on whether s'_i lies on the path $x, h(x), \ldots, h^{w-1}(x)$ or does not lie, with probability $1/2$ one of the problems $\texttt{2Pre}[h, w-1]$ or $\texttt{Inv}[h, w-1]$ is solved. □

Complex (and rather artificial) statements of the problems $\texttt{2Pre}[h, r]$ and $\texttt{Inv}[h, r]$ are explained by the need to take into account probabilistic distributions of intermediate keys $\text{WOTS}[H, h]$ for a random sk. A random choice of sk is supposed in $\texttt{EU-CMA}[\text{WOTS}[H, h]]$, we can not ignore it.

3.5.5 Shortening public keys

Let us discuss an additional technique used in our instantiation of WOTS and shortly mentioned in Section 1.3.2. This technique consists in replacing the lengthy public key $pk = (pk_1, pk_2, \ldots, pk_n)$ by a short one: $pk^* = h^*(pk_1, pk_2, \ldots, pk_n)$. Here h^* is an additional hash function $S^n \to R$. With this shortening, the verification equation takes the form:

$$h^*\left(h^{w-1-m_1}(s_1), \ldots, h^{w-1-m_n}(s_n)\right) \stackrel{?}{=} pk^*.$$

It is clear that the theorems above can be easily repaired to meet the new form of public key: it is sufficient to add the additional problem $\texttt{Coll}[h^*]$ to the stated compound problems.

Indeed, an adversary who attacks modified WOTS either solves the standard $\texttt{EU-CMA}[\text{WOTS}[H, h]](pk_1, pk_2, \ldots, pk_n)$ problem or finds different tuples $pk = (pk_1, pk_2, \ldots, pk_n)$ and $pk' = (pk'_1, pk_2, \ldots, pk'_n)$ such that $h^*(pk) = h^*(pk')$.

In our instantiation of WOTS in MAM2, we use $S = \mathbf{T}^{162}$, $w = 27$, $n = 81$, $R = \mathbf{T}^{243}$. We achieve that presumed Post-Quantum strengths of $\texttt{Coll}[H]$, $\texttt{Coll}[h^*]$ and the problems of the group $\{\texttt{Hook}[h, w-1], \texttt{2Pre}[h, w-1], \texttt{Inv}[h, w-1]\}$ are approximately the same:

$$w^{n/3} = |R|^{1/3} = |S|^{1/2} \approx 2^{128}.$$

For comparision, Pre-Quantum strengths of the same problems are approximately

$$w^{n/2} \approx 2^{192}, \quad |R|^{1/2} \approx 2^{192}, \quad |S| \approx 2^{256}.$$

Chapter 4

Formats

4.1 Rules

In this chapter we instantiate MAM2 expectations presented in Section 1.7. First, we convert expectations into a set of rules that cover interactions between MAM2 entities and objects which support these interactions. Second, we propose formats of target objects. To present the formats, we introduce a special data definition language called ProtoBuf3 (Section 4.2). It is based on the well-known Google Protocol Buffers notation. In Section 4.4 we discuss how to serialize ProtoBuf3 structures into ternary streams.

This chapter does not claim to be a detailed specification of MAM2 (the specification is attached to this report as a standalone document). It should be interpreted as an overview of the specification and as an explanation of chosen design decisions.

Let us start with rules. Some of our rules are also rules of MAM. We mark them with the label "MAM inherited".

Rule 1 (MAM inherited)**.** MAM2 messages are transmitted through the Tangle. Messages are divided into parts and these parts are put in the `sigMsgFragment` field of IOTA transactions (see Section 1.4 for details).

Rule 2. It is possible that the Tangle contains only headers of a message while the main message parts are transmitted outside the Tangle (so-called *detached messages* functionality). In this case the Tangle only announces the message and provides the information on how to decrypt it.

Rule 3 (MAM inherited)**.** Entities create *channels* (see Figure 4.1). Each entity can create an arbitrary number of channels.

Entities of MAM2 (as well as of IOTA and MAM) don't have strict identifiers. Therefore, the relations between channels and their owners are weak and non-transparent. This fact can be used to provide privacy and anonymity of channel's owners. Of course, the owner Alice can publish in the channel her own identification data if she wants.

Rule 4 (MAM inherited)**.** Each channel has a unique identifier called `chid` (channel id). It consists of 81 trytes and is put in the `address` field of the outer IOTA transactions.

Rule 5 (MAM inherited)**.** The identifier `chid` is a public key of MSS (Merkle Signature Scheme). This public key, called *root channel key*, covers a set of partial private / public keys called *leaf channel keys*.

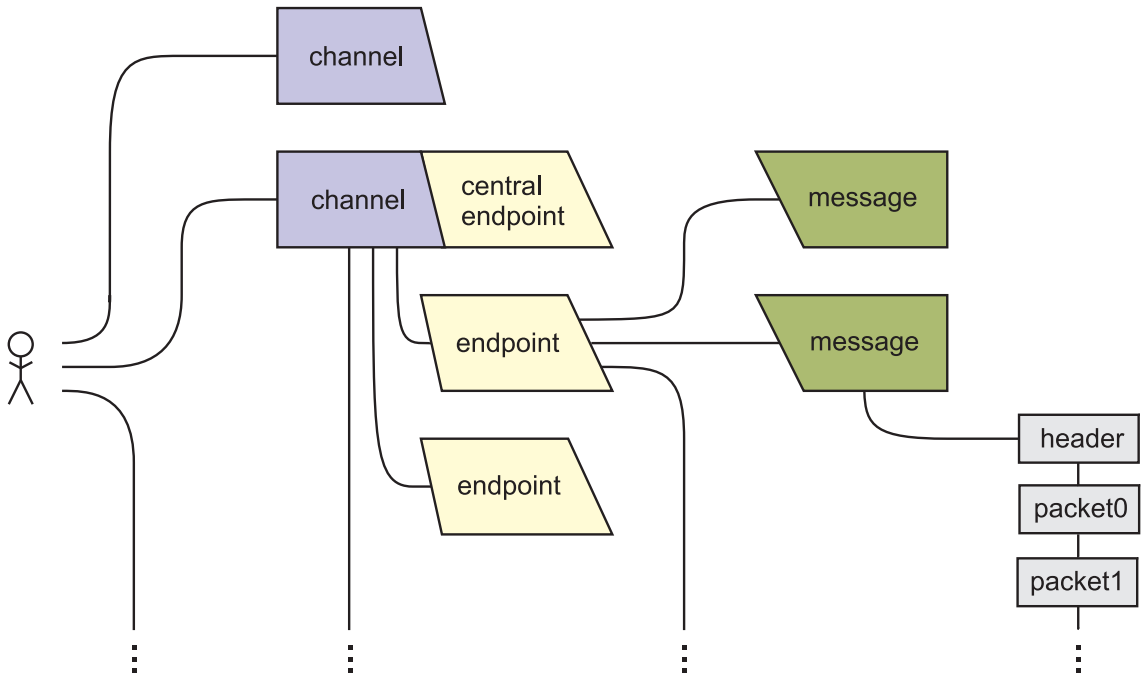

Figure 4.1: Channels, endpoints, messages

In MAM, channel keys are used to sign the messages transmitted through the channel. Unfortunately, Alice can sign only a restricted number of messages. It is due to inherent one-timeness of MSS: Alice cannot use a leaf private key twice (see Chapters 1 and 3 for details). Of course, after all leaf keys are run out, Alice can generate a new set of leaves, cover them with a new root `chid` and continue with it. Since `chid` unambiguously identifies the channel, Alice actually *changes the channel*. This approach is used in MAM.

The approach works but seems not to be very convenient. Indeed, after changing the channel Alice generally loses her subscribes and has to make considerable efforts to reattach them (or subscribers have to make considerable efforts to get a new Alice's `chid` and to attach to it).

We think that channels and therefore `chid` should be as permanent as possible. To support such functionality, we propose to use additional keys subordinate to `chid`. It could be multi-time keys of MSS or other type or it could be fully permanent keys.

We can imagine broadcasting stations which are equipped with subordinate keys and which help channels to transmit messages. Each station has a certain operating resource and a channel can launch new stations after exhausting the resources of active ones. We call such imaginary stations *endpoints*.

Rule 6. An owner of a channel creates an *endpoint* through which channel messages are transmitted. The owner can create an arbitrary number of endpoints in a given channel. An endpoint is equipped with a permanent public / private key pair or with several one-time key pairs (*leaf endpoint keys*). In case of several leaf keys they can be covered by a *root endpoint key*, for example, of MSS type. There always exists one implicit *central endpoint* which keys are channel keys.

In practical settings, an MSS root key can quite easily cover 2^{20} leaf keys. Therefore, a chain of two MSS root keys (channel and subordinate endpoint) covers 2^{40} leaves in total. If

1000000 channel messages (or their parts) are signed per day, then such amount of leaf keys is sufficient for approximately 3000 years.

Rule 7. Channel keys are used to sign endpoint keys. Endpoint keys are used to sign messages or their parts. Channel keys, interpreted as keys of the central endpoint, can be used to sign messages or their parts.

The pair (`chid`, a signed key) is an analogue of a conventional public key certificate.[1] In our quasi-certificate the `chid` field describes an issuer (a channel), the signed key describes a subject (an endpoint) and the signature describes delegation of trust from the issuer to the subject. Quasi-certificates are very laconic: they don't contain verbose identification data, validity periods and extensions with information about key's and subject's attributes.

A signed key is not necessarily an endpoint key. It could be also `chid` of another channel. In this case, continuing the analogy with certificates, we obtain a *cross-certificate*. Alice can use a cross-certificate if she still wants to change the channel.

Rule 8. A message is protected using a session (ephemeral) key K. This key is generated at random or pseudorandom for each message. It consists of 81 trytes. Protection of the message means applying AE algorithms.

The important point is that K is always fresh. Alice can adjust recipients of the specific message providing or not providing them information about K. For comparison, in MAM K is permanent. A recipient who once get K will be able to read all (future and past) messages of the channel.

Rule 9. A message is divided into *packets*. A transmitted packet consists of three parts: an ordinal number, an encrypted payload and a checksum. The payload is an informative part of the message. It is encrypted using a session key K. The checksum is optional. It is either a MAC generated using K or a signature of the MAC generated using an endpoint key.

Alice can manage sizes of the packets. For example, Alice can use only one packet for short messages and series of medium-sized packets for continuous video streams.

In MAM2, MACs are light: they are generated with minimal additional cost during message encryption and occupy insignificant space in output data streams. But signatures are usually heavy: they significantly prevalate over payloads in size and each their generation consumes a separate one-time leaf key. Fortunately, Alice can manage the issuance of checksums in the message packets. For example, she can put cheap MACs in every 10th packet and an expensive signature only in the last packet.

Signatures are generally necessary because they protect against a malicious recipient who knows K and issues packets with valid MACs from the Alice name. But in many cases, for example, in peer-to-peer relations with fully-trusted Bob, Alice can avoid signatures.

We thought that one `chid` can sign up to 2^{20} MSS endpoint keys (`epid`s), each `epid` can sign up to 2^{20} messages. Our refined estimates show that 2^{20} Merkle leaves are quite a lot, especially for lightweight IoT devices. We should restrict the number of leaves and therefore the total number of channel messages.

One approach to save a large total number is to use not one but several subordinate `epid`s: the ith `epid` is used to sign the $(i+1)$th one, the last `epid` is used to sign messages. This

[1] `https://en.wikipedia.org/wiki/Public_key_certificate`.

approach was extensively discussed in our team and, interestingly, has been presented recently in the paper [61]. A chain of `epid`s is very similar to a chain of conventional X.509 certificates.

We abandoned "A chain of epids" functionality despite the fact that it is quite easily encapsulated in MAM2. The main reason is that we hope that in future versions of MAM2 we will migrate to fully many-times signatures.

Rule 10. A packet's MAC and therefore signature covers the payload of this and all previous packets as well as identifiers of the current channel and endpoint.

Since a MAC covers all previous packets, it is not necessary to put an ordinal number of the packet under MAC control. Channel and endpoint dependence of MAC protects against transferring a message from one channel / endpoint to another.

Rule 11. A message is supplied with a *nonce*. Nonces of different messages of a given endpoint must differ. Therefore, a triple (channel, endpoint, nonce) unambiguously identifies a message.

Alice can use counters (for example, ordinal numbers of MSS leaf keys) as nonces.

Rule 12. A session key K is securely transmitted to groups of legal recipients in the form of *keyloads*. Each keyload contains full information about K and this information can be retrieved by only specified recipients. The standard keyloads are: Plain, PSK and PKE.

Different messages can be supplied with different keyloads. Thus, we can achieve a very flexible functionality: some messages of the channel are securely sent to one set of recipients, other messages to another.

Potentially, a keyload can be signed using an endpoint key. Such signing is the first step to build authenticated key establishment (AKE) and other cryptographic protocols. But for now such protocols are not necessary and we don't support signed keyloads.

Rule 13. In the Plain keyload, a session key K is presented in plaintext form. In means that the underlying message is actually not encrypted and that the packet MACs are actually intermediate or final hash values. But the message still can be authenticated by means of signatures.

If the Plain keyload is used, then message payloads are encrypted using a key that is known by everyone. It seems strange, but we decided to support just such functionality. The reason is that encryption in MAM2 isn't very expensive and it is very convenient to process both public and private messages in the same manner.

In presence of the Plain keyload all other keyloads should be avoided: they are trivially redundant.

The Plain keyload is actually used in MAM, where K is a public `chid`. Since K = chid is constant such approach doesn't provide a flexible functionality (see our comments on Rule 8).

Rule 14. In the PSK keyload, a session key K is presented in encrypted form. An encryption key is a *pre-shared key* (PSK) common for a group of recipients. The keyload contains an identifier of the group and the encrypted K. To encrypt K, the SE algorithms must be used. The output key stream (which is added to K during encryption) must differ for different message identification triples (channel, endpoint, nonce) even if the same PSK is used.

It is important to make the PSK encryption dependent on (channel, endpoint, nonce): an adversary can't use a key stream of one PSK keyload to reveal a key stream of another.

Rule 15. In the PKE keyload, a session key K is presented in encrypted form. An encryption key is a public key of a recipient.

The PKE keyload will be discussed in details in the next (Phase 2) report.

4.2 ProtoBuf3

4.2.1 Data definition languages

To describe formats of MAM2 messages, we should use some *data definition language* (DDL) oriented on non-text data. The main options are:

- Absract Syntax Notation One (ASN.1)[2];
- Google Protocol Buffers[3];
- TLS Presentation Language[4].

These languages have approximately equal capabilities: they all support common basic data types, enums, structs, choices, optional fields, etc.

We have chosen the middle option, Google Protocol Buffers v2, because it is simpler than the first option and richer than the last one. We refer to the chosen language as ProtoBuf. A specification of ProtoBuf can be found here: `https://developers.google.com/protocol-buffers/docs/proto`.

4.2.2 ProtoBuf

The building block of ProtoBuf is user-defined types marked with the `message` keyword.

Each type consists of *fields*. Each field has a name and a type. A field type can be either a simple type like `bool`, `string`, `bytes`, `uint32`, `double`, `enum` or a user-defined type.

Each field has a unique number which identifies the field within its type. A field number `n` is presented in the description of the field in the form "`= n`".

Fields are marked with *modifiers*. The main modifiers are:

- `required` — this field is obligatory;
- `optional` — this field is optional;
- `repeated` — this field can be repeated any number (including zero) of times;
- `oneof` — this field can be chosen from a given set of alternatives.

Here are several official examples that demonstrate main ProtoBuf constructions:

[2] `https://en.wikipedia.org/wiki/Abstract_Syntax_Notation_One`.
[3] `https://developers.google.com/protocol-buffers/`.
[4] `https://tools.ietf.org/html/rfc5246#page-7`.

```
message SearchRequest {
  required string query = 1;
  optional int32 page_number = 2;
  repeated int32 result_per_page = 3;
}

message SearchResponse {
  repeated Result result = 1;
}

message Result {
  required string url = 1;
  optional string title = 2;
  repeated string snippets = 3;
}

message SampleMessage {
  oneof test_oneof {
    string name = 1;
    SubMessage sub_message = 2;
  }
}
```

Note that the `Result` and `SearchResponse` types can be written also as follows:

```
message SearchResponse {
  message Result {
    required string url = 1;
    optional string title = 2;
    repeated string snippets = 3;
  }
  repeated Result result = 1;
}
```

4.2.3 Extending ProtoBuf: ProtoBuf3

To support the ternary logic of IOTA and MAM2, we propose a "ternary" extension of ProtoBuf. We call it ProtoBuf3. We will use ProtoBuf3 to describe MAM2 messages.

ProtoBuf3 uses additional to ProtoBuf data types and language constructions listed below.

- `null`: a special type that describes nothing and is encoded by the empty word;
- `tryte`: an element of $\mathbf{T}^3$. Interpreted as an integer, takes values in the range $[-13, 13]$;
- `trint`: an element of $\mathbf{T}^9$. Interpreted as an integer, consists of 3 trytes, takes values in the range $[-9841, 9841]$;

– `long trint`: an element of $\mathbf{T}^{18}$. Interpreted as an integer, consists of 6 trytes, takes values in the range $[-193710244, 193710244]$;

– `long long trint`: an element of $\mathbf{T}^{27}$. Interpreted as an integer, consists of 9 trytes, takes values in the range $[-3812798742493, 3812798742493]$;

– `trytes`: an array of trytes. The length of the array is implicitly encoded before the array elements;

– `T arr[n]`: an array `arr` of `n` elements of type `T`;

– `T arr[]`: an array `arr` of elements of type `T`. The array can be placed only at the end of a data object. Elements of `arr` are continued until the end of the object. The number of elements is not fixed during encoding, it is determined indirectly during decoding. Note that this field does not have any modifiers.

In Protobuf3, we disable the `optional` modifier. A construction like

```
optional T something;
```

can be replaced by

```
oneof something_or_nothing {
  null nothing = 0;
  T something = 1;
}
```

Disabling `optional`, we need to wrap `optional` into `oneof` but, on the other hand, reduce a number of syntactic constructions and related encoding rules. In this case, we prefer simplicity of flexibility. After disabling `optional`, the `required` modifier becomes uninformative, we also disable it.

Note that optional functionality is implicitly presented in such constructions as `trytes arr` or `T arr[]`. An underlying array `arr` can be empty and, therefore, optional.

MAM2 messages are processed using cryptographic algorithms: some parts of messages are hashed, other parts are encrypted, etc. The main processing is based on a sponge function F (see Section 1.3.3) and AE algorithms (Chapter 2). We can imagine a ternary stream which includes plaintext fragments, public headers, keys, external objects, etc. Portions of the stream are sequentially absorbed in the sponge state. The state is squeezed out from time to time with further transformation of plaintexts into ciphertexts and generation of MACs. It would be convenient to describe the semantic of processing portions, which are presented as ProtoBuf3 fields, using the following cryptographic modifiers:

– `absorb` — this field is absorbed into the sponge state. We usually absorb keys, nonces, public headers;

– `squeeze` — this field is squeezed from the state. We usually squeeze MACs and pseudo-random numbers;

– `crypt` — this field is encrypted or decrypted;

– `skip` — this field must not be processed cryptographically; We usually skip signatures and duplicate objects;

– `external` — this field is an external object. It can be absorbed or squeezed by `spongos` although not being presented in the resulting ternary stream.

The above modifiers cannot be assigned to fields of user-defined types.

A field can have only one of the modifiers `absorb`, `squeeze`, `crypt` and `skip`. The `external` modifier can be combined with any of them.

If no one of the modifiers `absorb`, `crypt`, `squeeze` and `skip` is assigned to a field explicitly, then `absorb` is assigned implicitly.

Two additional modifiers control a sponge state:

– `fork` — call $\texttt{spongos}' \leftarrow \texttt{spongos.Fork}(\perp)$. All fields after this call and until the end of the current user-defined type must be processed using $\texttt{spongos}'$. The `spongos` object should still be used to process the main message stream;

– `commit` — force $\texttt{spongos.Commit}(\perp)$. Usually used immediately after absorbing a key or nonce.

Actually, `fork` and `commit` are sponge commands. But we assume for coherence that they are modifiers of empty (`null`) fields.

To reduce the size of transmitted messages, we discard the field numbers used in ProtoBuf. For example, the ProtoBuf line

```
trytes query = 1;
```

turns into

```
trytes query;
```

We leave field numbers only in `oneof` constructions where they are necessary. Note that the field numbers take values in the range $[-13, 13]$.

In ProtoBuf, a `oneof` could not have an additional modifier. In ProtoBuf3, this possibility exists.

When describing fields in ProtoBuf3, one should adhere to the following considerations: the cryptographic modifiers are the first to be written, then other modifiers (`oneof`, `repeated`), then a type of the field, then its name.

4.2.4 IOTA transactions in ProtoBuf3

Let us demonstrate how to describe the format of IOTA transactions[5] using ProtoBuf3.

```
message Transaction {
  tryte signatureMessageFragment[2187];
  tryte address[81];
  long long trint value;
  tryte obsoleteTag[27];
  trytes timestamp;
  trint currentIndex;
  trint lastIndex;
```

[5] https://docs.iota.org/introduction/iota-token/anatomy-of-a-transaction.

```
    tryte bundle[81];
    tryte branchTransaction[81];
    tryte trunkTransaction[81];
    tryte attachmentTag[27];
    trint attachmentTimestamp;
    trint attachmentTimestampLowerBound;
    trint attachmentTimestampUpperBound;
    tryte nonce[27];
}
```

4.3 Data types

4.3.1 The `Msg` type

A MAM2 message is described by the following ProtoBuf3 type:

```
message Msg {
  Channel channel;
  Endpoint endpoint;
  Header header;
  Packet packets[];
}
```

The fields of `Msg` have the following meaning:

- `channel` — a description of a channel to which the message belongs;
- `endpoint` — a description of an endpoint which is used to transmit the message. If this field is absent then the central channel endpoint (equipped with `chid`) is used;
- `header` — a header of the message. The header contains keyloads for recipients that allow recovering a session key K which has been used to protect the message;
- `packets` — message packets.

It isn't necessary that all parts of the message are presented in one data unit (for example, in a IOTA bundle). The parts can be scattered over several units and these units can be even outside the Tangle.

The important point is that packets don't contain the identifiers of the message to which they belong. So, in the case of scattered packets, recipients should know where they can find the packets and how they have to combine them. Hints for recipients are ordinal numbers and checksums contained in packets.

4.3.2 The `Channel` type

A MAM2 channel is described by the following type:

```
message Channel {
  tryte ver;
  external tryte chid[81];
}
```

The fields of `Channel` have the following meaning:

- `ver` — a version of MAM2. The current version is 0;
- `chid` — an identifier of the target channel. The `external` modifier says that the identifier isn't actually presented in the `Channel` container: it is put in the `address` field of the outer bundle (see Section 1.4). But even an implicit `chid` is processed during hashing of the target message as if it is really presented.

4.3.3 The `Endpoint` type

A MAM2 endpoint is described by the following type:

```
message Endpoint {
  oneof pubkey {
    null chid = 0;
    tryte epid[81] = 1;
    SignedId chid1 = 2;
    SignedId epid1 = 3;
  }
}
message SignedId {
  tryte id[81];
  MSSig mssig;
}
message MSSSig {
  external squeeze tryte mac[78];
  skip trytes sig;
}
```

The `pubkey` field of the `Endpoint` type describes a public key that can be used to verify signatures of endpoint messages or their parts. It could be either:

- `chid` — a key of this channel;
- `chid1` — a key of another channel;
- `epid` and `epid1` — a subordinate endpoint key.

In the cases `chid1` and `epid1`, a public key is accompanied with a signature. The unsigned `epid` key must be signed somewhere earlier, that is, it must be published in the `epid1` field of some previous message in this channel.

The signed public key is described by the `SignedId` type, where `id` is a public key and `mssig` is a signature. The signature must be generated using the `MSS` layer under the private key that corresponds to `chid`.

The syntax of `MSSig` means that to generate the signature `sig` one must first squeeze `mac` and then sign it.

The `Channel` and `Endpoint` parts of a message form an object called a quasi-certificate in Section 4.1. These parts have independent meaning and the message may not contain the remaining parts. For example, if Alice wants to change her channel she publishes a quasi-certificate with a new `chid` in the `Endpoint` part and finishes the message at this point.

4.3.4 The `Header` type

A message header is described by the following types:

```
message Header {
  tryte nonce[27];
  repeated oneof keyload {
    KeyloadPlain plain = 0;
    KeyloadPSK psk = 1;
    KeyloadNTRU ntru = 2;
  }
  external tryte key[81];
  commit;
}

message KeyloadPlain {
    fork;
    tryte key[81];
}
message KeyloadPSK {
    fork;
    tryte id[27];
    external tryte psk[81];
    commit;
    crypt tryte ekey[81];
}
message KeyloadNTRU {
    fork;
    tryte id[27];
    tryte ekey[3072];
}
```

The fields of `Header` have the following meaning:

- `nonce` — a unique message nonce;
- `keyloads` — a tuple of keyloads of different types;
- `key` — a session key K that is implicitly inserted in the message stream. The insertion makes a sponge state secret and allows to generate MACs and produce ciphertexts in subsequent packets.

Each keyload starts with the `fork` modifier. It means that cryptographic processing of keyload fields is performed in a branch of the main message stream. In particular, MACs or signatures of the main stream do not control the keyload data.

The `KeyloadPlain` type describes the Plain keyload. This keyload reveals a session key so that the target message can be decrypted by anyone although its integrity and authenticity control is still possible. The `key` field of `KeyloadPlain` is the session key K in the plain form. If the Plain keyload is included in a header, then all other keyloads become redundand and should be avoided.

The `KeyloadPSK` type describes the PSK (Pre-Shared Key) keyload. This keyload contains a session key encrypted using a previously delivered PSK. The `id` field presents an identifier of a group of recipients who share the same PSK key. The `psk` field presents this key, and the `ekey` field presents the encrypted K.

The `KeyloadNTRU` type describes the NTRU keyload (see 5.6) that supports PKE (Public Key Encryption). This keyload contains a session key encrypted using recipient's public key. The `NTRU` layer and, more precisely, the `NTRU.Encr` algorithm is used for encryption. The `id` field of `KeyloadNTRU` presents first trytes of recipient's public key and the `ekey` field presents the encrypted K.

4.3.5 The `Packet` type

A message packet is described by the following type:

```
message Packet {
  long trint ord;
  crypt trytes payload;
  oneof checksum {
    null none = 0;
    squeeze tryte mac[81] = 1;
    MSSig mssig = 2;
  }
  commit;
}
message MAC {
}
```

The fields of `Packet` have the following meaning:

- `ord` — an ordinal number of the packet (zero-based numbering). The choice of the `long trint` type means that a message can contain at most $193710244 \approx 2^{27}$ packets. We think that it is enough for all practical purposes (taking into account that packets can be arbitrary large). And it much much less than the limits on the amount of data that can be processed on a single session key K (see Chapter 2);
- `payload` — an informative part of the message. It is encrypted using a session key K;
- `checksum` — some control characteristic of the current packet data as well as previous packets and message headers. It could be a MAC (the `mac` field), a signature of the MAC (`mssig`) or even the empty field (`none`).

Note that in packets we avoid an explicit identification of the parent message (in particular, we don't include in the packet a message nonce) because we want the packets to be as short as possible.

4.3.6 Settings

There are the following `oneof` and implicitly optional fields of the `Msg` type (see Figure 4.2, colored rectangles):

- `endpoint.pubkey` (`oneof`);
- `endpoint.sig` (implicitly optional);
- `header.keyload` (`oneof`);
- `packets` (implicitly optional);
- `packets[i].checksum` (`oneof`).

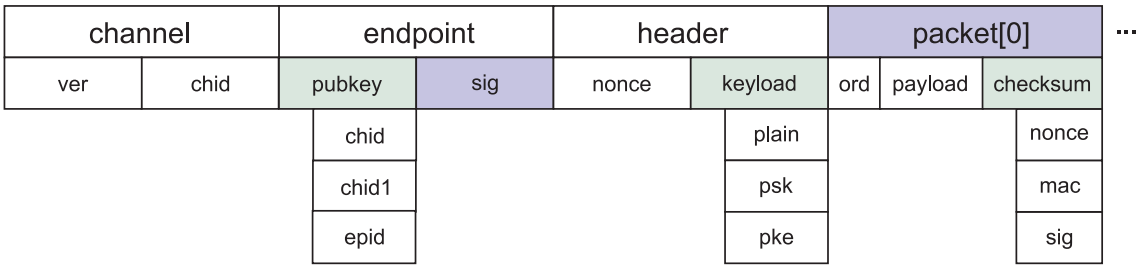

Figure 4.2: A message

In the following table, we present important settings for the first fields from the list above.

Table 4.1: Settings for `Msg` fields

`endpoint`		`header`	Meaning
`pubkey`	`sig`	`keyload`	
`chid`	–	`plain`	A public channel (MAM inherited, packets are signed using `chid`)
`chid`	–	`psk`	A restricted channel (MAM inherited)
`chid1`	+		Announcing a new channel (MAM inherited)
`epid1`	+		Launching a new endpoint (subordinate keys which will be used to sign packets)
`epid`	–		Using an existing endpoint

4.4 Encoding conventions

Data structures with defined fields are encoded: they are represented as strings of trytes. In this section, we describe the rules for encoding of basic types (`null`, `tryte`, `trint`, etc.), composite types (`trytes`, `T arr[n]`, `T arr[]`), modifiers (`repeated`, `oneof`) and `message`. These rules ensure the correct parsing of the tryte string to separate fields.

In comparison with ProtoBuf, in ProtoBuf3 the encoding rules undergo significant changes. This is due to two factors:

– the use of other types of data;

– it is necessary that, after encoding, the message is increased to the minimum possible length.

4.4.1 Encoding of service data

Sometimes, when encoding, we need to specify the number of field repetitions or the length occupied by some type. For this, we use the service data type `size_t`. This type is an integer that takes only non-negative values.[6]

We have considered several variants of this type representation.

1. The type `size_t` is encoded as a trytes string, which consists of 2 consecutive parts: `code` and `data`.

   The part `data` is an integer that occupies the minimum possible number of trytes. The size (in trytes) of this field is specified in the `code`.

   The part `code` takes 1 tryte. It takes values in the range $[0, 13]$.

   Thus, type `size_t` takes values in the range $[0, 2026277576509488133]$.

   For example, the number `153` will be encoded as `BRF` (`code`: `B`; `data`: `RF` $= -9 \cdot 27^0 + 6 \cdot 27^1$).

2. The type `size_t` is encoded by a tryte string as a number with base 14. This means that all trytes except for the last one take values in the interval $[0, 13]$. Instead of the value of the last tryte, which takes values in the interval $[1, 13]$, we write the opposite of it (that is, from the interval $[-13, -1]$). The negative tryte is the end of coding marker.

   For example, the number `153` will be encoded as `MQ` $(13 \cdot 14^0 - (-10) \cdot 14^1)$.

3. The type `size_t` is encoded by a trint string as a number with base 9842. This means that all trints except for the last one, take values in the interval $[0, 9841]$. Instead of the value of the last trint, which takes values in the interval $[1, 9841]$, we write the opposite of it (that is, from the interval $[-9841, -1]$). The negative trint is the end of coding marker.

   For example, the number `153` is encoded as `IU9`: $-(9 \cdot 27^0 - 6 \cdot 27^1 + 0 \cdot 27^2) \cdot 9842^0$.

However, both the second and third variants have two significant drawbacks:

– uneconomical encoding of large numbers;

[6] To represent this and other integer fields the little-endian conventions are used. This means that the first tryte is the least significant tryte, and the last one — the most significant tryte. For example, the number `1001` will be encoded as `BJA` $(2 \cdot 27^0 + 10 \cdot 27^1 + 1 \cdot 27^2)$.

– it is impossible to encode a zero value.

That's why it was decided to use variant 1 of the type `size_t` for encoding.

4.4.2 Encoding of basic types

Encoding of `null`. A `null` field is encoded as an empty string.

Encoding of integers. Fields of the types `tryte`, `trint`, `long trint`, `long long trint` are encoded by 1, 3, 6, 9 trytes respectively. The little-endian conventions are used.

For example, the number `20000` of type `long trint` is encoded as `TL9A99`: $-7 \cdot 27^0 + 12 \cdot 27^1 + 0 \cdot 27^2 + 1 \cdot 27^3 + 0 \cdot 27^4 + 0 \cdot 27^5$.

4.4.3 Encoding of composite types

Encoding of `trytes`. A field of type `trytes` is encoded as a string, which consists of 2 sequentially concatenated parts: `size_t` and `data`.

The `data` part is a tryte array that has a length (in trytes), specified in the `size_t` part.

For example, an array `SOMETHINGARRAY` will be encoded as `BNASOMETHINGARRAY` (`size`: `BNA`; `data`: `SOMETHINGARRAY`).

Encoding of `T arr[n]`. The array `T arr[n]` consists of the concatenation of sequentially encoded elements `arr[0], arr[1], ..., arr[n-1]`.

For example, a field `tryte something[3]`, when `arr[0] = X`, `arr[1] = Y` and `arr[2] = Z`, will be encoded as `XYZ`.

Encoding of `T arr[].` An array `T arr[n]` consists of the concatenation of sequentially encoded elements `arr[0]`, `arr[1]`,... until the encoded data object runs out. The number of elements is not fixed in the encoding process, it is determined indirectly during decoding.

4.4.4 Encoding of modifiers

Encoding of `oneof`. The `oneof` modifier is encoded as a tag of type `tryte`, in which the field number is located (from the set of possible for this `oneof` fields). After the tag, there is a field corresponding to the specified number.

For example, for the field `test_oneof`, if you select an option `name` = `QWERTY` we receive the encoded message `9AFQWERTY`.

```
oneof test_oneof {
  trytes name = 0;
  SubMessage sub_message = 1;
}
```

Encoding of `repeated`. The `repeated` modifier is encoded as a tag of type `size_t`, which determines the number of repetitions of this field. After the tag, an encoded field is located, repeating (one by one) the number of times specified in the tag.

For example, decoding the field `repeated tryte len` whose value is `ACABC`, we get that this field is repeated 3 times, taking the values `1`, `2` and `3` respectively.

Absorbing `oneof` alternatives and `repeated` repetitions. In the first version of Protobuf3, we didn't absorb cryptographically `oneof` alternatives and `repeated` repetitions. This situation can raise security issues. Consider, for example, the following type:

```
oneof file_status {
    trytes file_ok = 0;
    trytes file_not_found = 1;
}
```

A file status is set through one of the `file_status` alternatives (0 or 1), and a file name is set using the `file_ok` or `file_not_found` fields. The problem is that the selected alternative is not absorbed, although it is presented in an encoded stream. Manipulating the stream, an adversary can change the alternative keeping cryptographic checksums valid. Such drawback is unacceptable, and we decided to absorb selected alternatives along with the corresponding fields.

Similar problems arise around `repeated` repetitions which we also decided to absorb.

4.4.5 Encoding of `message`

A code of a message consists of the concatenation of sequentially encoded fields. If one message is nested in another message, the encoding is performed recursively. The order of the fields must strictly correspond to the order of the fields in the message format.

Chapter 5

Public Key Encryption Algorithms

5.1 Concept

5.1.1 PKE

Public Key Encryption (PKE) allows encrypting a *plaintext* using a public key of a recipient and decrypting the resulting *ciphertext* using the corresponding private key. Usually, a plaintext is a session key that is used for symmetric encryption of target data (a *payload*). PKE is not used for direct encryption of payloads because known PKE systems are much slower than their symmetric counterparts. It turns out to be natural and effective to encrypt potentially large payloads using fast symmetric algorithms under a volatile session key and after that encrypt these keys using slow PKE algorithms. Such an approach, known as *hybrid encryption*, is used in MAM2.

PKE is defined by three algorithms: `Gen`, `Encr`, `Decr`.

The `Gen` algorithm takes a security parameter l (canonically, in the form of the unary code 1^l) and returns a private key sk and a public key pk:

$$\texttt{Gen}\colon 1^l \mapsto (sk, pk).$$

The entity who runs `Gen` keeps sk in secret and publishes pk.

The parameter l usually determines lengths of keys and texts. The greater l, the longer keys and, correspondingly, the harder to attack them but, simultaneously, the slower the resulting algorithms.

There could be an additional algorithm `Setup` that takes 1^l and returns long-term parameters shared by many entities:

$$\texttt{Setup}\colon 1^l \mapsto params.$$

These parameters are used in `Gen` instead of 1^l:

$$\texttt{Gen}\colon params \mapsto (sk, pk).$$

`Setup` can be even not an algorithm but a procedure or approach used by inventors of the PKE system. The inventors can choose *params* very carefully and fix them in a specification of the system.

We will use both interfaces of `Gen`. The latter interface will be more common. Let us suppose for simplicity that optional *params* are included into pk.

The `Encr` algorithm takes a plaintext pt, a public key pk and returns a ciphertext ct:

$$\texttt{Encr}\colon (pt, pk) \mapsto ct.$$

Conversly, `Decr` takes a ciphertext ct, a pair (sk, pk) of keys and returns a plaintext pt or the special symbol $\perp$:

$$\texttt{Decr}\colon (pt, sk, pk) \mapsto pt \text{ or } \perp.$$

The returning of $\perp$ means that inputs are invalid (for example, ct is corrupted). The key pk is used as input because it can contain *params* or other information useful for decryption.

`Gen` is always probabilistic, `Encr` is usually probabilistic, `Decr` is usually deterministic.

Logically, it must hold that

$$\texttt{Decr}(\texttt{Encr}(pt, pk), sk, pk) = pt$$

for each (sk, pk) generated with `Gen` and each valid pt. But some PKE systems allow so-called *decryption failures* when the latter equation can be violated with a small probability.

Example 5.1 (ElGamal)**.** *In the textbook ElGamal cryptosystem,* ***`Setup`*** *generates a cyclic group G of prime order q. Plaintexts and ciphertexts are elements of G. The group G is described by one of its generator g.*

Other ElGamal algorithms are:

Algorithm Gen	**Algorithm** Encr	**Algorithm** Decr
Input: g.	*Input:* (pt, y).	*Input:* (ct, x).
Steps:	*Steps:*	*Steps:*
1. $x \xleftarrow{R} \{0, 1, \ldots, q-1\}$.	1. $k \xleftarrow{R} \{0, 1, \ldots, q-1\}$.	1. *Represent* ct *as* (u, v).
2. $y \leftarrow g^x$.	2. $ct \leftarrow (g^k, pt \cdot y^k)$.	2. $pt \leftarrow v/u^x$.
3. *Return* x *and* y.	3. *Return* ct.	3. *Return* pt.

Here x and y are private and public keys respectively. Note that ***`Gen`*** *and* ***`Encr`*** *are indeed probabilistic. Note also that* ***`Decr`*** *does not return $\perp$: all ciphertexts from $G \times G$ are allowed.*

5.1.2 KEM

A lightweight alternative to PKE is Key Encapsulation Mechanism (KEM). The main difference is that a sender doesn't control a plaintext. As we said before, this plaintext is usually a session key and it is rather reasonable that the key can be generated internally during its encryption. Now we are talking not about *encryption* but about *encapsulation*.

In KEM, the `Encr` algorithm is changed to `Encap`. It takes a public key pk and returns a ciphertext ct with an encapsulated (and internally generated) *key*:

$$\texttt{Encap}\colon pk \mapsto (ct, key).$$

Since `Encap` must generate key, this algorithm is necessary probabilistic. A sender saves key and sends ct.

A recipient runs the reverse algorithm `Decap`. It has the following interface:

$$\texttt{Decap}\colon (ct, sk, pk) \mapsto key \text{ or } \perp.$$

There exist an easy way to convert KEM into PKE using hybrid encryption. Let Bob want to send Alice, an owner of (sk, pk), a plaintext pt. Then Bob:

1) runs $\texttt{Encap}(pk)$ and gets (ct, key);

2) encrypts pt using symmetric crypto under key and gets additional ciphertext ct';

3) sends a pair (ct, ct') interpreted as a ciphertext of PKE.

Alice:

1) runs $\texttt{Decap}(ct, sk, pk)$ and gets $\perp$ or key;

2) in the case of $\perp$, interrupts decryption returing $\perp$;

3) decrypt ct' under key and gets pt.

We see that differences between KEM and PKE schemes can be overcome, more precisely, KEM can be extended to PKE. Nevertheless, we will give preference to PKE. We need PKE in MAM2 (where the same session key must be sent to many recipients) and it would be better if we provide "clean" PKE, not an extension of KEM.

5.1.3 Security

An adversary who attacks a PKE system $(\texttt{Gen}, \texttt{Encr}, \texttt{Decr})$ knows pk that has been generated using `Gen`. Therefore, the adversary has full access to `Encr`, that is, she can encrypt any plaintext she wants. Such attack conditions are usually attributed as *Chosen Plaintext Attack* (CPA).

The adversary can also have access to the `Decr` algorithm providing some ciphertexts and getting either the corresponding plaintexts or $\perp$. New conditions are called *Chosen Ciphertext Attack* (CCA). Note that at any moment the adversary can choose any ciphertext she wants. So we do not differentiate between non-adaptive (all ciphertexts are chosen before some stage of the attack) and adaptive (often denoted by CCA2) attacks.

The main problems for the adversary are the following:

1. Determine sk.

2. Determine randomly chosen pt by the corresponding $ct = \texttt{Encr}(pt, pk)$.

3. Given $ct = \texttt{Encr}(pk, pt_b)$ find $b \xleftarrow{R} \{0, 1\}$. Here pt_0 and pt_1 are different plaintexts that the adversary chooses her own after obtaining pk.

The first problem is the hardest, the last one is the easiest. In the last two problems, the query to `Decr` with ct is not allowed (using such a query, the adversary trivially solves the target problem).

Resistance to attacks for solving the 2nd problem is called *one-wayness* (OW). Usually, this abbreviation is prolonged with the type of attack: OW-CPA or OW-CCA. Resistance regarding the 3rd problem is called *indistinguishability* (IND). Extended abbreviations are IND-CPA and IND-CCA.

Example 5.2. *ElGamal is OW-CPA if the CDH problem in G is hard. Let us remind that CDH (Computational Diffie-Hellman) stands as follows: Given*

$$(g, g^x, g^k : x, k \xleftarrow{R} \{0, 1, \ldots, q\})$$

find g^{xk}. If ElGamal isn't OW-CPA then an adversary can determine pt from $(g, y, g^k, pt \cdot y^k)$. After that, the adversary can determine y^k. In other words, she finds g^{xk} from (g, g^x, g^k), that is, solves DDH.

Example 5.3. *ElGamal is IND-CPA if the DDH problem in G is hard. Let us remind that the DDH (Decisional Diffie-Hellman) problem stands as follows: Given*

$$(g, g^x, g^k, g^s : x, k \xleftarrow{R} \{0, 1, \ldots, q-1\})$$

decide if $s \equiv xk \pmod q$ (answer 1) or s is chosen at random from $\{0, 1, \ldots, q-1\}$ (answer 0). If ElGamal isn't IND-CPA then an adversary A can distinguish between $(g^k, pt_0 \cdot g^{xk})$ and $(g^k, pt_1 \cdot g^{xk})$. Then another adversary B can solve DDH using A as a subroutine. Given (g, g^x, g^k, g^s), B acts as follows:

1) *choses arbitrary $pt_0, pt_1 \in G$;*

2) *generates $b \xleftarrow{R} \{0, 1\}$;*

3) *sends to A the plaintexts pt_0, pt_1 and the ciphertext $(g^k, pt_b \cdot g^s)$;*

4) *returns 0 if A did not guess b and 1 otherwise.*

If s was chosen at random, then A processes a random ciphertext and cannot get an advantage in determining b. So B returns 1 with probability 1/2. But if $s \equiv xk \pmod q$, then A works in fair ElGamal settings and determines b with non-negligible advantage. In this case B returns 1 with probability 1/2 plus this advantage. In result, A's advantage becomes B's advantage, that is, B solves DDH.

Example 5.4. *The textbook ElGamal cryptosystem is not secure in CCA settings. It is due to the following fact: if $\mathtt{Encr}(pt_b, pk) = (u_b, v_b) = (g^{r_b}, pt_b \cdot y^{r_b})$, $b = 0, 1$, then*

$$(u_0, v_0) \cdot (u_1, v_1) = (u_0 \cdot u_1, v_0 \cdot v_1) = \left(g^{k_0+k_1}, pt_0 \cdot pt_1 \cdot y^{k_0+k_1}\right)$$

can be interpreted as a result of encryption of $pt_0 \cdot pt_1$. Consequently,

$$\mathtt{Decr}(\mathtt{Encr}(pt_0, pk) \cdot \mathtt{Encr}(pt_1, pk), sk) = pt_0 \cdot pt_1.$$

An adversary can use this indentity to hide decryption of any ciphertext in CCA settings. Indeed, if the adversary is not allowed to decrypt ct_0, then she generates a random pt_1, encrypts it and obtains ct_1, then decrypts $ct_0 \cdot ct_1$ and gets $pt_0 \cdot pt_1$. Since the adversary knows pt_1, she can reconstruct pt_0.

The previous examples show that we cannot automatically derive OW-CCA or IND-CCA properties from their OW-CPA or IND-CPA counterparts.

But there exists an easy way to achieve OW-CCA or even IND-CCA starting from OW-CPA. This way was proposed by Fujisaki and Okamoto in [90] and is called the *Fujisaki — Okamoto* (FO) *transform*.

The FO transform utilizes the idea of hybrid encryption wrapping in ct an ephemeral key σ and accompanying ct with a result of symmetric encryption of pt under σ. The main point here is that σ fully determines random coins used during the PKE encryption. Alice repeats the encryption and checks that the result is indeed ct. If the validation fails, then `Decr` returns $\perp$. Due to this logic, an adversary cannot impose arbitrary data for decryption.

The similar construction, called NTRU Asymmetric Encryption Padding (NAEP), was proposed in [116] as a part of the NTRU cryptography (see 5.2.6).

5.1.4 PKE and DS

The very perspective idea is to use the same long-term parameters or even keypairs (sk, pk) both in the PKE and DS algorithms. In the case of combining parameters, we can reuse basic (usually arithmetic) algorithms. In the case of combining keys, the benefits are even greater: we can reduce the amount of protected and usually rather expensive memory for storing private keys, we can simplify the distribution of public keys, we can organize the proof-of-knowledge of PKE private keys self-signing them.

Let us recall that the proposed DS algorithms (see Chapter 3) belong to the HBC platform. Unfortunately, this platform in principle does not allow constructing PKE or KEM schemes.

Fortunately, other platforms discussed below contain prerequisites for the dual usage of parameters and even keys. The prerequisites but no concrete solutions. It is a challenge for us to combine PKE and DS "all rolled into one".

A well-known cryptographic principle says that "a key must be used only for one purpose". In the case of combining, this principle is violated and we have to carefully analyze possible security drawbacks.

The nowadays standard is EUF-CMA for DS and IND-CCA for PKE. During analysis we must show that

1) if the DS keys are used in the PKE algorithms, then DS still possesses EUF-CMA;

2) if the PKE keys are used in the DS algorithms, then PKE still possesses IND-CCA.

This analysis is quite individual, it crucially depends on target schemes.

Example 5.5. *Let ElGamal/PKE keys (x, y) are also used in the Schnorr/DS algorithms. In these algorithms, a hash function $\varphi\colon G \times \boldsymbol{T}^* \to \{0, 1, \ldots, q-1\}$ is additionally used.*

The DS algorithms look as follows:

Algorithm Sign

Input: $m \in \boldsymbol{T}^*$ *(a message),* x.
Steps:

1. $k \xleftarrow{R} \{0, 1, \ldots, q-1\}$.
2. $r \leftarrow g^k$.
3. $e \leftarrow \varphi(r, m)$.
4. $s \leftarrow (k - xe) \bmod q$.
5. *Return* (e, s) *(a signature).*

Algorithm Verify

Input: m, (e, s), y.
Steps:

1. *Check a format of* (e, s). *Return 0, if the format is invalid.*
2. *Check if* $\varphi(g^s \cdot y^e, m) = e$. *Return* 0, *if the equation does not hold.*
3. *Return* 1.

Suppose that ElGamal/PKE is supported by the FO transform. Then all queries to `Decr` *return* $\perp$ *except for queries that contain valid tuples* (g, g^x, g^k, g^{xk}). *In other words, the adversary has additional access to the DDH oracle. The Schnorr DS is attacked with an additional oracle. Intuitively, there is no reason to believe that this oracle can facilitate attacks to DS. But intuition must be supported by rigorous reasonings.*

An adversary who attacks DS obtains additionally tuples (m, r, s). *Again there is no reason to believe that they can be used to reduce the strength of PKE. Again we should provide rigorous reasonings.*

5.1.5 NPQCC

In Chapter 1, we have mentioned NIST Post-Quantum Crypto Competition (NPQCC) and the main cryptoplatforms to which belong its participants (see Table 1.2).

At the first stage of the development of PKE algorithms, we carefully analyzed these platforms. We were looking for the most suitable platform.

Since NPQCC provides the latest information on the subjects, we actively used it. More precisely, we compared platforms through their best representatives which are the participants of NPQCC.

During the comparison, we answered the following questions:

1. How short keys and ciphertexts can be?
2. How fast the PKE algorithms can be?
3. How much can we trust the platform given the history of cryptoanalytic attacks?
4. Can we construct DS algorithms using PKE long-term parameters?
5. Can we construct DS algorithms using PKE keypairs?
6. If we can construct suitable DS algorithms, how fast they can be and how short their signatures can be?
7. How appropriate is the ternary logic?

The results of the analysis are presented in Sections 5.2 — 5.5. To reference candidates of NPQCC, we accompany their names with the prefix "NPQCC/": NPQCC/NTRUEncrypt, NPQCC/LAKE, etc. Specifications of the candidates and related information could be found here: `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions`.

5.2 Lattice-based schemes

5.2.1 Introduction

Let us start with a problem of the Fourth International Student' Olympiad in Cryptology NSUCRYPTO [103].

Problem (FNV2)**.** The FNV2 hash function is derived from the function FNV-1a (`http://www.isthe.com/chongo/tech/comp/fnv/`). FNV2 processes a message x composed of bytes $x_1, x_2, \ldots, x_n \in \{0, 1, \ldots, 255\}$ in the following way:

1) $h \leftarrow h_0$;

2) for $i = 1, 2, \ldots, n$: $h \leftarrow (h + x_i)g \bmod 2^{128}$;

3) return h.

Here $h_0 = 144066263297769815596495629667062367629$, $g = 2^{88} + 315$. Find a collision, that is, two different messages x and x' such that $\mathrm{FNV2}(x) = \mathrm{FNV2}(x')$. □

The problem can be easily solved using the famous LLL algorithm [138]. Consider the vectors

$$\begin{aligned}
\mathbf{b}_1 &= (1, 0, 0, \ldots, 0, g^{n-2} \bmod 2^{128}),\\
\mathbf{b}_2 &= (0, 1, 0, \ldots, 0, g^{n-3} \bmod 2^{128}),\\
&\ldots\\
\mathbf{b}_{n-1} &= (0, 0, 0, \ldots, 1, g^0 \bmod 2^{128}),\\
\mathbf{b}_n &= (0, 0, 0, \ldots, 0, t2^{128}),
\end{aligned}$$

where t is some small nonzero integer. The vectors $(\mathbf{b}_i)$ are linearly independent (over $\mathbb{R}$). They form a matrix B which ith row is $\mathbf{b}_i$.

If we multiply all possible integer vectors by B we get the set

$$\mathcal{L}(B) = \mathbb{Z}^n B = \{\mathbf{z}B \colon \mathbf{z} \in \mathbb{Z}^n\}$$

called a *lattice*.

Let us recall some basic concepts and facts related to lattices.

1. The vectors $\mathbf{b}_i$ and, therefore, matrix B can be real-valued. We avoid this case because it is never used in cryptography.

2. Vectors of $\mathbb{R}^n$ and therefore $\mathcal{L} \subset \mathbb{R}^n$ are equipped with a norm to become a metric space. The norm $\|\mathbf{v}\|$ of a vector $\mathbf{v} \in \mathbb{R}^n$ is also called its *length*. The usual norm is Euclidian:

$$\|(v_1, v_2, \ldots, v_n)\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2},$$

also called L_2. The Euclidian norm can be generalized changing the constant 2 above by an integer $p \geqslant 3$. In other contexts we will also use the maximum norm (L_∞):

$$\|(v_1, v_2, \ldots, v_n)\|_\infty = \max_i |v_i|.$$

We mean L_2 in $\|\mathbf{v}\|$ unless otherwise stated.

3. Let $R^{m\times n}$ be a set of all matrices of dimensions $m \times n$ over a set R. Another generalization is to allow B to be non-square. That is, B belongs to $\mathbb{Z}^{m\times n}$ and consists of m rows $\mathbf{b}_i \in \mathbb{Z}^n$. Here $m \leqslant n$ to approve linear independence of $\mathbf{b}_i$. We suspend this generalization for a while. The dimension m is called a *rank* of the lattice $\mathbb{Z}^m B$.

4. The lattice $\mathcal{L}(B)$ is an additive group, the subgroup of $\mathbb{Z}^n$ or even $\mathbb{R}^n$. Indeed, if $\mathbf{u}, \mathbf{v} \in \mathcal{L}(B)$, then $\mathbf{u} \pm \mathbf{v} \in \mathcal{L}(B)$. The group $\mathcal{L}(B) \subset \mathbb{R}^n$ is *discrete*: each its element doesn't contain other elements in an enough small but non-negligible neighborhood. Conversly, any discrete additive subgroup of $\mathbb{R}^n$ is a lattice (see [110]).

5. The matrix B is a *basis* of $\mathcal{L}(B)$. The absolute value of B's determinant is a *determinant* of the lattice: $\det \mathcal{L} = |\det B|$.[1] Hadamard's inequality yields: $\det \mathcal{L} \leqslant \prod_{i=1}^{n} \|\mathbf{b}_i\|$. The upper bound is tight if the basis vectors $\{\mathbf{b}_i\}$ are pairwise orthogonal. The determinant $\det \mathcal{L}$ is also called a *volume* of $\mathcal{L}$.

6. The inverse determinant characterizes the density of the lattice vectors in $\mathbb{Z}^n$. More precisely, $|\mathbb{Z}^n/\mathcal{L}|$, the number of cosets of $\mathbb{Z}^n$ modulo $\mathcal{L}$, equals $\det \mathcal{L}$ and one of the $\det \mathcal{L}$ points of $\mathbb{Z}^n$ belongs to $\mathcal{L}$.

7. Denote by $\lambda_i(\mathcal{L})$ the *ith minimum of* $\mathcal{L}$, that is, the smallest radius r such that the ball $\{\mathbf{x} \in \mathbb{Z}^n\colon \|\mathbf{x}\| \leqslant r\}$ contains i linearly independent vectors of $\mathcal{L}$.

8. *Hermit's constants* $\gamma_1, \gamma_2, \ldots$ are the smallest values such that

$$\lambda_1(\mathcal{L}) \leqslant \sqrt{\gamma_n} (\det \mathcal{L})^{1/n}$$

for every $\mathcal{L}$ of dimension n. Only $\gamma_1, \ldots, \gamma_8$ and γ_{24} are known. Hermite's theorem yields that $\gamma_n \leqslant n$. Moreover, for a rather large n its holds that $\gamma_n \in [n/(2\pi e), n/(\pi e)]$, where π and e are the usual universal constants.

9. The basis B can be replaced by another basis $B^* \in \mathbb{R}^{n\times n}$ such that

$$\mathcal{L}(B) = \mathcal{L}(B^*).$$

[1] If B is non-square, then $\det \mathcal{L} = \sqrt{\det BB^T}$.

If $B^* \in \mathbb{Z}^{n\times n}$, then the transformation is described by a unimodular matrix U of order n:

$$B^* = UB.$$

Let us recall that a unimodular matrix consists of integers and its determinant is ± 1. So $V = U^{-1}$ is also unimodular and in the previous equation B and B^* can be swapped:

$$B = VB^*.$$

Note that $|\det B| = |\det B^*|$ and $\det \mathcal{L}$ is independent of the choice of the basis.

"Good" bases that consist of *short* (with respect to some metric) and almost orthogonal vectors are usually interesting. The mentioned LLL algorithm finds a "quite good" basis B^* given arbitrary basis B.

Returning to the FNV2 problem, the basis B is "bad" because it contains very large numbers in its last column. Suppose that $B^* = \mathrm{LLL}(B)$ contains a rather short row $\mathbf{a} = (a_1, \ldots, a_{n-1}, a_n)$ such that $-255 \leqslant a_1, \ldots, a_{n-1} \leqslant 255$ and $a_n = 0$. Since $\mathbf{a} \in \mathcal{L}(B)$, we have

$$(a_1, \ldots, a_{n-1}, a_n) = \sum_{i=1}^{n-1} a_i \mathbf{b}_i + \alpha \mathbf{b}_n$$

for some integer α. But it implies that

$$\sum_{i=1}^{n-1} a_i g^{n-2} \equiv 0 \pmod{2^{128}}.$$

Let us represent each a_i in the form $a_i = x_i - x'_i$, $x_i, x'_i \in \{0, 1, \ldots, 255\}$. Now $\mathrm{FNV2}(x_1 \ldots x_{n-1}) = \mathrm{FNV2}(x'_1 \ldots x'_{n-1})$ because the difference between hash values is

$$\big((h_0 + x_1)g^{n-1} + x_2 g^{n-2} + \ldots + x_{n-1} g\big) - \big((h_0 + x'_1)g^{n-1} + x'_2 g^{n-2} + \ldots + x'_{n-1} g\big) \equiv$$
$$\equiv \sum_{i=1}^{n-1} a_i g^{n-i} \equiv 0 \bmod 2^{128}.$$

It remains to say that the proposed technique allows finding collisions starting from $n = 18$ (further we will explain why). To apply LLL, one can use almost any existing computer algebra system — LLL is an almost mandatory component of such systems. And, importantly, LLL runs very fast, we could find solutions to the FNV2 problem almost instantly.

We have started talking about the LBC platform with the easy example related to cryptography. Based on this example, we have introduced lattices, discussed their "good" and "bad" bases, demonstrated the effectiveness of the LLL algorithm.

We do not aspire to the completeness of the presentation of the mathematical background of LBC, it is the topic of individual articles and even books. Our task is to convey the basic ideas in a maximally simple form. Other useful information and links to further reading can be found, for example, in [110, 150, 159].

5.2.2 Hard problems

The main idea of LBC is the following: a private key is a "good" basis B^* of some lattice and a public key is a "bad" basis B of the same lattice. Since $B = VB^*$, B is a "masked" copy of B^*. We will encounter a similar "matrix masquerading" in the CBC and MBC cryptoplatforms (see Sections 5.3, 5.4), it is a common place in public key cryptography.

Alice keeps B^* in secret and publishes B. Using B, Bob encrypts plaintexts for Alice or verifies her signatures. Using B^*, Alice decrypts and signs. Usually, knowledge of B^* allows Alice to solve hard problems related to the lattice $\mathcal{L} = \mathcal{L}(B) = \mathcal{L}(B^*)$. These problems are hard for an adversary who doesn't know B^*.

Three classical hard problems are the following.

Problem (Shortest Vector Problem, SVP)**.** Given B, find a nonzero $\mathbf{v}^* \in \mathcal{L}(B)$ such that $\|\mathbf{v}^*\| = \lambda_1(\mathcal{L})$. □

Problem (Shortest Independent Vectors Problem, SIVP)**.** Given B, find linearly independent $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^* \in \mathcal{L}(B)$ such that $\max_i \|\mathbf{v}_i^*\| = \lambda_n(\mathcal{L})$. □

Problem (Closest Vector Problem, CVP)**.** Given B and $\mathbf{x} \in \mathbb{Z}^n \setminus \mathcal{L}(B)$, find $\mathbf{v} \in \mathcal{L}(B)$ closest to $\mathbf{x}$: $\|\mathbf{v} - \mathbf{x}\|$ is as small as possible. □

In 1981 Van Emde Boas proved NP-hardness of SVP in the L_∞ norm and CVP in the L_p, $p \geqslant 3$, and L_∞ norms. In 1998 Miklós Ajtai proved that SVP in Euclidian norm is also NP-hard for randomized reductions (that is, there is a probabilistic Turing-machine which in polynomial time reduces any problem in NP to instances of SVP). These results make up the foundation of LBC.

There are some variations of the introduced problems:

1. CVP with the promise that $\|\mathbf{x} - \mathbf{v}\| \leqslant \frac{1}{2}\lambda_1(\mathcal{L})$ is known as Bounded Distance Decoding (BDD).

2. Covering Radius Problem (CRP) is to compute

$$\rho(\mathcal{L}) = \max_{\mathbf{x} \in \mathbb{Z}^n} \min_{\mathbf{v} \in \mathcal{L}} \|\mathbf{x} - \mathbf{v}\| .$$

3. The SVP problem can be relaxed: given B and $\gamma > 1$, find nonzero $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| \leqslant \gamma\lambda_1(\mathcal{L})$. The relaxed problem is called *Approximate-SVP* and is denoted by SVP_γ. CVP and SIVP have similar approximate versions: CVP_γ and SIVP_γ.

4. The SVP problem can be relaxed in other manner: given B and $\gamma > 1$, find nonzero $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| \leqslant \gamma(\det \mathcal{L})^{1/n}$. This problem is called *Hermite-SVP*: the factor γ obviously relates to Hermit's constants.

5. All the problems above are search problems. They have decision counterparts. For example, *Decision SVP* is to decide if a given $\mathbf{v}$ is the shortest vector of a lattice $\mathcal{L}(B)$ generated by a given B.

Many reductions between the introduced problems are known. Let us recall that a problem P_1 *reduces* to a problem P_2 if an algorithm A_2 which solves P_2 can be converted into a

polynomial algorithm A_1 which solves P_1 using A_2 as an oracle. The reduction is *randomized* (see above) if A_1 is probabilistic and its success probability is non-negligible. The reduction is *quantum* (see below) if A_1 is quantum.

In [102] it was proposed a nice reduction from SVP to CVP. It turns out that to solve SVP(B) it is sufficient to solve n instances CVP($B^{(i)}, \mathbf{b}_i$), $i = 1, 2, \ldots, n$, in which $B^{(i)}$ is obtained from B by doubling its ith row. If $\mathbf{v}_i$ is an answer to CVP($B^i, \mathbf{b}_i$), then the shortest vector of the set $\{\mathbf{b}_i - \mathbf{v}_i\}$ is the answer to SVP(B). Indeed, let $\mathbf{v} = \sum_i c_i \mathbf{b}_i$ be the shortest vector of $\mathcal{L}(B)$. Then

1) there exists j such that c_j is odd (otherwise, $\mathbf{c}/2$ belongs to $\mathcal{L}(B)$ and is shorter than $\mathbf{c}$);

2) $\mathbf{u} = \frac{1}{2}(c_j + 1)(2\mathbf{b}_j) + \sum_{i \neq j} c_i \mathbf{b}_i$ belongs to $\mathcal{L}(B^{(j)})$;

3) $\mathbf{u} - \mathbf{b}_j = \mathbf{v}$,

from which the required result follows.

Other known reductions:[2]

- SVP_γ reduces to CVP_γ;
- $\text{CVP}_{\gamma'}$ reduces to SVP_γ, where $\gamma' = \text{poly}(\gamma, n)$;
- SVP reduces to computing λ_1;
- CVP reduces to computing $\min_{\mathbf{v} \in \mathcal{L}} \|\mathbf{x} - \mathbf{v}\|$.

5.2.3 Hardness

It is proved that the main approximate problems $\texttt{SVP}_\gamma$, $\texttt{CVP}_\gamma$, $\texttt{SIVP}_\gamma$ are NP-hard for $\gamma = n^{O(1/\log\log n)}$. If $\gamma = O(\sqrt{n})$, then these problems belong to the intersection NP $\cup$ coNP and for $\gamma = 2^{\Omega(n)}$ they can be solved using the polynomial algorithms like LLL. [3] Cryptography exploits the range between small and large γ. The zone $\gamma = n^{O(1)}$ is usual for cryptographic applications. We call this zone *moderate*.

Is this zone safe? What we can say about the hardness of target problems with concrete, not asymptotic, values of n and γ? What is about hardness on the average, not in the worst case? These questions were intensively studied in [92].

Gama and Nguyen, the authors of [92], made experiments with random lattices $\mathcal{L} \subset \mathbb{R}^n$ (they proposed a special technique for their generation). Note that for a random $\mathcal{L}$, the approximations

$$\frac{\lambda_i(\mathcal{L})}{(\det \mathcal{L})^{1/n}} \approx \sqrt{\frac{n}{2\pi e}}, \quad i = 1, 2, \ldots, n,$$

hold with overwhelming probability. The first approximation is known as the *Gaussian heuristic*.

[2] See `https://www.iacr.org/workshops/tcc2007/Micciancio.pdf`, `https://cseweb.ucsd.edu/classes/sp07/cse206a/lec7.pdf`.

[3] $f(n) = \Omega(g(n))$ if $g(n) = O(f(n))$.

Example 5.6. *Let us return to the FNV2 problem. Suppose that the underlying lattice $\mathcal{L}$ looks like random. We are interested in vectors $\mathbf{v} = (v_1, \ldots, v_{n-1}, 0) \in \mathcal{L}$ with coordinates $v_i \in [-255, 255]$. For these vectors, $\|v\| \leqslant 255\sqrt{n-1}$, and we can roughly determine the smallest possible n from the inequality*

$$\lambda_1(\mathcal{L}) \approx \sqrt{\frac{n}{2\pi e}} \cdot 2^{128/n} \leqslant 255\sqrt{n-1}.$$

Here we use the Gaussian heuristic and the fact that $\det \mathcal{L} = 2^{128}$. Our rough estimate is $n' = 13$. It is much less than the smallest dimension $n'' = 18$ provided by LLL in our experiments. The difference between n' and n'' demonstrates limited capabilities of LLL. □

In [92] several polynomial algorithms for finding a small basis $\{\mathbf{b}_i^*\}$ of a random $\mathcal{L}$ were applied. One of the algorithms was LLL which finds a so-called *reduced basis*. Its important feature is that

$$\min_i \|\mathbf{b}_i\| \leqslant 2^{n-1}\lambda_1(\mathcal{L}),$$

the fact that we will use soon.

The quality of each algorithm was characterized by the *Hermite factor* $\min_i \|\mathbf{b}_i^*\| / (\det \mathcal{L})^{1/n}$. For random lattices, it is close to the approximate factor $\gamma = \min_i \|\mathbf{b}_i^*\| / \lambda_1(\mathcal{L})$ achievable by the target algorithm and, theorefore, expresses the loss factor of the length of the shortest output vector relative to the shortest vector of the lattice.

It is proved that the Hermite factor of LLL isn't greater than

$$\delta \leqslant \left(\frac{4}{3} + \varepsilon\right)^{(n-1)/4},$$

where ε is small. This estimate is valid for any basis and is tight.[4] The estimate nicely relates to Hermit's inequality:

$$\gamma_n \leqslant \gamma_2^{n-1} = \left(\frac{4}{3}\right)^{(n-1)/2}.$$

Usually, the Hermite factor is written as δ^n. In the experiments of Gama and Nguyen, δ did not fall below 1.011. The citation from [150]: «*Lower values of δ seem to be impossible to obtain with our current understanding of lattice reduction. Gama and Nguyen in fact estimate that a factor of 1.005 is totally out of reach in dimension 500*».

The very important thing is that *there is no quantum algorithms that approximates lattice problems to within polynomial factor.* This statement is also from [150] where it was formulated as conjecture shared by the majority of researchers in the field. Nevertheless, on a quantum computer problems are solved slightly faster. So in necessary cases, we will provide both classic and quantum estimates of security.

Summing up, the moderate approximate problems with random instances, which are usual in cryptography, are believed hard to solve by known algorithms including quantum ones. Let us emphasize that we are talking about *random* instances. If lattices we used have some hidden structure, then security of overlying cryptosystems can dramatically fall. Our next example demonstrates this fact.

[4] Experiments shows that $4/3 + \varepsilon \approx 1.33$ can be replaced by 1.08 (with $\delta = 1.019$, see below) if we deal with random lattices.

Example 5.7 (knapsacks). *Knapsack cryptosystems were very popular in the late 1970s. The main idea is very simple. Let $r_1, r_2, \ldots, r_n$ be a superincreasing integer sequence:*

$$r_i > r_1 + \ldots + r_{i-1}, \quad i = 2, \ldots, n,$$

with a rather large $r_1 > 2^n$ (to provide security against easy attacks, see [153]). Knowing this sequence, Alice can easily solve the equation

$$S = \sum_{i=1}^{n} pt_i r_i$$

in (pt_i), bits of a plaintext. In order to make the solution difficult, Alice masks (r_i) by chosing coprime A and $M > 2r_n$ and replacing r_i with

$$R_i = Ar_i \bmod M.$$

Alice publishes (R_i) and keeps (r_i), A, M in secret. To encrypt pt, Bob calculates $ct = \sum_{i=1}^{n} pt_i R_i$. Alice restores $S = A^{-1} \cdot ct \bmod M$ using (A, M) and reconstructs (pt_i) using (r_i). An adversary who doesn't know secrets has to find a short vector in an $(n+1)$-dimensional lattice $\mathcal{L}$ induced by the following basis vectors:

$$\begin{aligned} \mathbf{b}_1 &= (1, 0, \ldots, 0, R_1), \\ \mathbf{b}_2 &= (0, 1, \ldots, 0, R_2), \\ &\ldots \\ \mathbf{b}_n &= (0, 0, \ldots, 1, R_n), \\ \mathbf{b}_{n+1} &= (0, 0, \ldots, 0, S). \end{aligned}$$

This lattice contains the desired vector

$$\mathbf{pt} = \sum_{i=1}^{n} pt_i \mathbf{b}_i - \mathbf{b}_{n+1} = (pt_1, pt_2, \ldots, pt_n, 0),$$

which is very short: its norm doesn't exceed $\sqrt{n}$. The lattice looks like random and it seems to be very hard to find such a small vector using LLL or a similar algorithm.

Unfortunately, it is not true. Due to the superincreasing property,

$$M > 2r_n > 4r_{n-1} > \ldots > 2^{n+1} r_1 > 2^{2n+1}$$

and $R_i = O(2^{2n})$, $S = O(2^{2n})$. Therefore, the density $n/\max(R_1, \ldots, R_n, S)$ is close to $1/2$ what is very small comparatively to random lattices. Such a small density yields that with an overwhelming probability (see [135] for details):

- *short vectors of $\mathcal{L}$ are either collinear to $\mathbf{pt}$ or their norms are greater than $2^{n-1} \|\mathbf{pt}\|$;*
- *$\lambda_1(\mathcal{L}) = \|\mathbf{pt}\|$.*

Since LLL returns a vector $\mathbf{b}_i^* \in \mathcal{L}$ which norm doesn't exceed $2^{n-1}\lambda_1(\mathcal{L})$, one can find $\mathbf{pt}$ in polynomial time applying LLL several times.

Although the knapsack initial basis looks very similar to the FNV2 one, these bases are very different. For example, the density of FNV2 is $n/128$ and it grows with n. □

5.2.4 q-ary lattices and LWE

In cryptography, we cannot use keys-matrices with arbitrary large entries from $\mathbb{Z}$ because we need to store and transmit these keys. So the entries are reduced modulo medium-large integer q. Technically, the lattice $\mathcal{L}$ is made *q-ary*: $q\mathbb{Z}^n \subseteq \mathcal{L} \subseteq \mathbb{Z}^n$. Now the membership $\mathbf{v} \in \mathcal{L}$ is completely determined by $\mathbf{v} \mod q$. The q-arness is easily achieved by extending B with the block qI_n. Such an extension means that we go from $\mathcal{L}(B)$ to the lattice

$$\Lambda_q(B) = \{\mathbf{v} \in \mathbb{Z}^n \colon \mathbf{v} \equiv \mathbf{z}B \bmod q \text{ for some } \mathbf{z} \in \mathbb{Z}_q^n\}.$$

Usually $\Lambda_q(B)$ is accompanied with the dual (up to normalization) lattice

$$\Lambda_q^\perp(B) = \{\mathbf{v} \in \mathbb{Z}^n \colon \mathbf{v}B^T \equiv 0 \pmod q\}.$$

Note that the matrix B is not a basis of $\Lambda_q(B)$ nor $\Lambda_q^\perp(B)$, but these bases can be easily derived from B using linear algebra.

We can easily switch to the lattices generated by a non-square $B \in \mathbb{Z}_q^{m\times n}$, $m \leqslant n$. For example,

$$\Lambda_q(B) = \{\mathbf{v} \in \mathbb{Z}^n \colon \mathbf{v} \equiv \mathbf{z}B \bmod q \text{ for some } \mathbf{z} \in \mathbb{Z}_q^m\}.$$

It is interesting that if $B \in \mathbb{Z}_q^{m\times n}$ has full rank, then $\Lambda_q^\perp(B)$ contains exactly q^{n-m} vectors from $\mathbb{Z}_q^n$ from which it follows that $\det(\Lambda_q^\perp(B)) = q^m$ (the inverse of density of $\Lambda_q^\perp(B)$ in $\mathbb{Z}_q^n$).

The important fact is that $\Lambda_q^\perp(B)$ always contains a vector of length q. It could be, for example, the vector $(q, 0, \ldots, 0)$. This situation is common for q-ary lattices. So if we want to hide in such a lattice a short vector, its length must be significantly less than q. It is indeed the fact and lengths of hidden vectors are usually of order $\sqrt{n} \ll q$.

SVP in $\Lambda_q^\perp(B^T)$ is related to the Short Integer Solution (SIS) problem: given B, find short $\mathbf{x}$ such that $\mathbf{x}B \equiv 0 \pmod q$.

Dealing with q-ary lattices, we will usually use the *centered* representation of $\mathbb{Z}_q$, that is, $\mathbb{Z}_q = \mathbb{Z} \cap [-q/2, q/2) = \{-\lfloor q/2 \rfloor, \ldots, \lfloor q/2 \rceil - 1\}$ where $\lfloor z \rceil = \lfloor z + 1/2 \rfloor$. We can imagine that an implicit modular reduction operator $\bmod q$ is changed to $\operatorname{mods} q$. (The $\operatorname{mods} q$ operator can be easily adapted to even q.)

An additional important problem often used in LBC is Learning With Errors (LWE). It was introduced by Regev in [167]. Its formulation is rather complicated, so we need an additional formalism.

Let $\xleftarrow{\chi}$ denote the random choice of a tuple of objects (for example, a vector from coordinates) such that each object is independent of other and conforms a distribution χ. For example, R in $\xleftarrow{R}$ means the uniform distribution.

Let χ be a known probabilistic distribution over $\mathbb{Z}_q$ (typically the discrete Gaussian, see 5.2.8). The inputs to LWE are a matrix $A \xleftarrow{R} \mathbb{Z}_q^{n\times m}$ and a vector $\mathbf{b} \in \mathbb{Z}_q^m$ which is either chosen uniformly or has the form

$$\mathbf{b} = \mathbf{s}A + \mathbf{e},$$

where $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$ and $\mathbf{e} \xleftarrow{\chi} \mathbb{Z}_q^m$. It is required to distinguish with some non-negligible probability between these two cases. In other words, we have to distinguish between random and "structured" vectors $\mathbf{b}$.

The important point is that χ "produces" small vectors. They stand as errors which mask the "signal" $\mathbf{s}A$. That is why *learning with errors*. It will be convenient to write

$$\mathbf{b} \overset{\chi}{\approx} \mathbf{s}A$$

or even $\mathbf{b} \approx \mathbf{s}A$ if χ is clear from the context.

We have formulated the so-called *Decision-LWE*. In *Search-LWE*, it is required to find $\mathbf{s}$ given A and $\mathbf{b}$. Trivially, Decision-LWE reduces to Search-LWE. The non-trivial fact proved in [167] is that Search-LWE reduces to Decision-LWE when q is a prime bounded by some polynomial in n.

Search-LWE can be naturally reformulated if we replace the matrix A by an oracle that sequentially reveals columns of A along with the corresponding coordinates of $\mathbf{b}$.

Problem (Search-LWE)**.** Let O be an oracle which is instantitated with $\mathbf{s} \in \mathbb{Z}_q^n$ and outputs the pairs

$$(\mathbf{a}, b = (\mathbf{s}\mathbf{a}^T + e) \bmod q), \quad \mathbf{a} \overset{R}{\leftarrow} \mathbb{Z}_q^n, \quad e \overset{\chi}{\leftarrow} \mathbb{Z}_q.$$

Given n, q, χ and an access to O, find $\mathbf{s}$.

Let Ψ_α denote the rounded Gaussian distribution on $\mathbb{Z}_q$ with mean 0 and standard deviation $\alpha q/\sqrt{2\pi}$, $\alpha q > \sqrt{n}$. In [167] it was found a quantum reduction of $\mathrm{SIVP}_{\tilde{O}(n/\alpha)}$ and Decision $\mathrm{SVP}_{\tilde{O}(n/\alpha)}$ to Decision LWE with $\chi = \Psi_\alpha$.[5] This reduction is a strong argument for hardness of LWE.

The important thing which significantly increases guarantees of the hardness of LWE is its random self-reducibility: an algorithm A that solves a large fraction of the instances can be converted with polynomial expenses to an algorithm B that solves all instances with an overwhelming probability. Indeed, if we are talking about Search-LWE, then instances are vectors $\mathbf{s}$. The algorithm B having access to the oracle O equipped with $\mathbf{s}$ generates $\mathbf{t} \overset{R}{\leftarrow} \mathbb{Z}_q^n$ and simulates for A an oracle $O_{\mathbf{t}}$ which changes O's answers from $(\mathbf{a}, b)$ to $(\mathbf{a}, b + \mathbf{t}\mathbf{a}^T)$. In other words, B enforces A to solve LWE with the instance $\mathbf{s}' = \mathbf{s} + \mathbf{t}$. Trying several random t, B rather quickly finds an instance $\mathbf{s}'$ which A can solve and from which B can recover $\mathbf{s} = \mathbf{s}' - \mathbf{t}$.

Random self-reducibility means that the average-case complexity of LWE is the same as the worst case complexity.

5.2.5 Parameters

From matrices to polynomials. To describe keys, that is, bases $B, B^* \in \mathbb{Z}_q^{n \times n}$, it is necessary to provide n^2 elements of $\mathbb{Z}_q$. It could be a problem because safe keys usually require rather large n and the total amount of key material would be unsatisfactory huge. In LBC to overcome this drawback, structured matrices are used. The chosen structure allows to effectively compress matrices. In fact, it is sufficient to save only their first rows.

The main idea is to represent matrices by polynomials. Let $\phi(x) \in \mathbb{Z}[x]$ be monic polynomial of degree n. It induces the factor ring $R = \mathbb{Z}[x]/(\phi(x))$ consisting of polynomials with integer coefficients which degrees are less than n. If we reduce coefficients mods q, we get polynomials from the ring $R_q = \mathbb{Z}_q[x]/(\phi(x))$.

[5] $\tilde{O}(x)$ means $O(x \log x)$.

Let $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$ be some vector of $\mathbb{Z}_q^n$. Associate it with the polynomial $a(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1}$. Then derive additional polynomials

$$x^{i-1}a(x) \bmod \phi(x)$$

and convert each of them into a coefficient vector $\mathbf{a}_i$, $i = 2, \ldots, n$. Finally, combine $\mathbf{a}, \mathbf{a}_2, \ldots, \mathbf{a}_n$ into a matrix A.

In fact, we interpret a vector $\mathbf{a}$ first as the polynomial $a(x)$ and second as the matrix A. The resulting matrix is indeed structured: given $\phi(x)$ we only need to know its first row. Another important point is that if A' is another such matrix derived from a polynomial $a'(x)$, then the sum $A + A'$ and product AA' correspond to the polynomials $a(x) + a'(x)$ and $a(x)a'(x)$ respectively. In other words, there exists an isomorphism from the polynomial ring R to the reflected set of matrices.

Further we will interchangeably use both polynomial (small letters, $a(x)$ or just a) and matrix notations (capital letters, A). In rare cases, we will also use vector notations (bold small letters, $\mathbf{a}$).

A polynomial $a(x) \in R_q$ is *small* if the corresponding vector $\mathbf{a} \in \mathbb{Z}_q^n$ is short. Here use the centered representation of the set $\mathbb{Z}_q$ to which belong coefficients of $a(x)$ or coordinates of $\mathbf{a}$. We tend to use the centered representation everywhere further. Naturally, $\|a\| = \|\mathbf{a}\|$.

The relation between polynomials and matrices becomes very clear if $\phi(x) = x^n - 1$. In this case, A is a *circulant* derived from $\mathbf{a}$:

$$A = \mathrm{circ}(\mathbf{a}) = \begin{pmatrix} a_0 & a_1 & a_2 & \ldots & a_{n-1} \\ a_{n-1} & a_0 & a_1 & \ldots & a_{n-2} \\ & & \ldots\ldots\ldots & & \\ a_1 & a_2 & a_3 & \ldots & a_0 \end{pmatrix}.$$

The well-known circulant is the identity matrix $I_n = \mathrm{circ}(1, 0, \ldots, 0)$.

Let us derive B from a random $b(x) \in \mathbb{Z}_q[x]/(\phi(x))$ and build lattices $\mathcal{L} = \Lambda_q(B)$ and $\mathcal{L} = \Lambda_q^{\perp}(B)$. Since B has a structure, its distribution differs from the uniform over $\mathbb{Z}_q^{n \times n}$. There exists a danger that $\mathcal{L}$ possesses some structural properties which can facilitate solving lattice-related problems. Fortunately, intensive experiments show that under reasonable choice of $\phi(x)$ the lattice $\mathcal{L}$ is hardly distinguished from "really random". So we can quite confidently use such lattices in LBC.

RLWE and MLWE. Lattice-related problems are transformed in the transition from matrices to polynomials. For example, the LWE problem becomes RLWE (Ring-LWE). Its search form has the following form.

Problem (Search RLWE)**.** Let O be an oracle which is instantitated with $s \in R_q$ and outputs the pairs

$$(a, sa + e), \quad a \xleftarrow{R} R_q, \quad e \xleftarrow{\chi} R_q.$$

Given descriptions of $R_q = \mathbb{Z}_q[x]/(\phi(x))$, χ and access to O, find s.

As in LWE, a polynomial e generated according to χ tends to be small. It stands as an error or noise which hides the signal sa.

Let us focus on the important difference between the LWE and RLWE problems. In LWE, the "signal objects" $\mathbf{s}$ and $\mathbf{a}$ are from the *module* $\mathbb{Z}_q^n$, they are multiplied using the inner product

$$\mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_q, \quad (\mathbf{s}, \mathbf{a}) \mapsto \mathbf{s}\mathbf{a}^T.$$

The "noise object" e is from $\mathbb{Z}_q$. While in RLWE, both signal and noise objects are from the *ring* R_q, signal objects are multiplied using standard multiplication in R_q.

Interestingly, we can build a module R_q^k over R_q. Elements of this module are vectors which coordinates belong to R_q. They are naturally multiplied again using the inner product:

$$R_q^k \times R_q^k \to R_q, \quad (\mathbf{s}, \mathbf{a}) \mapsto \mathbf{s}\mathbf{a}^T.$$

From the algebraic point of view, nothing happened. We just change $\mathbb{Z}_q$ with R_q.

We can easily reformulate LWE for the new algebraic construction. The adapted LWE is called MLWE (Module-LWE).

Problem (Search MLWE)**.** Let O be an oracle which is instantitated with $s \in R_q^k$ and outputs the pairs

$$(\mathbf{a}, \mathbf{s}\mathbf{a}^T + \mathbf{e}), \quad \mathbf{a} \xleftarrow{R} R_q^k, \quad \mathbf{e} \xleftarrow{\chi} R_q.$$

Given descriptions of $R_q^k = (\mathbb{Z}_q[x]/(\phi(x)))^k$, χ and an access to O, find $\mathbf{s}$.

The use of MLWE in cryptography was first proposed by the authors of NPQCC/KYBER. They motivate their proposal in two aspects. First, Alice can fix R_q, speed up arithmetic in this ring and then manipulate the module dimension k to achieve a required security. Second, several attacks are known for which it is important to use modules of dimension $k \leqslant 2$. In fact, NTRU uses modules of dimension 2 (see 5.2.6).

We discuss hardness of RLWE and MLWE in 5.2.7 when representing encryption schemes that built upon these problems.

The polynomial $\phi(x)$. There are the three main options for the polynomial $\phi(x)$: $\phi(x) = x^n - 1$, $\phi(x) = x^n + 1$, and $\phi(x) = x^n - x - 1$.

The first option is the most conservative, it was used in the first (*classic*) versions of the NTRU cryptosystem (see 5.2.6). The security of induced lattices is dramatically fallen if n is composite [95]. So $\phi(x) = x^n - 1$ is always used with prime n.

The second option $\phi(x) = x^n + 1$ allows to effectively apply NTT (Number Theoretic Transform), the discrete version of FFT (Fast Fourier Transform). Using NTT, we can significantly speed up multiplication and division of polynomials from $\mathbb{Z}_q[x]/(\phi(x))$.[6]

Let us briefly overview NTT. To apply the transform in the most convenient manner, n should be a power of two: $n = 2^s$. We assume that it is the case.[7] Looking ahead, let $\mathbb{Z}_q$ contain an element γ of multiplicative order $2n$ and $\omega = \gamma^2$ be the element of order n. If $f(x) = f_0 + f_1 x \ldots + f_{n-1}x^{n-1}$ is some polynomial over $\mathbb{Z}_q$, then NTT(f) is a polynomial $\hat{f}(x) = \hat{f}_0 + \hat{f}_1 x + \ldots + \hat{f}_{n-1}x^{n-1}$ with the coefficients

$$\hat{f}_j = \sum_{i=0}^{n-1} \gamma^i f_i \omega^{ij} \mod q, \quad j = 0, 1, \ldots, n-1.$$

[6] For comparison, to acheive a competitive speed up of polynomial arithmetic with $\phi(x) = x^n - 1$, the NTRU authors proposed to use so-called *product form* polynomials. Unfortunately, the idea of "product form" is covered by a patent which will expire only in 2021 (see [171] for details). It is interesting that NPQCC/NTRUEncrypt, which is submitted by the same authors, doesn't use product form polynomials. Fortunately, in 2017, the patent was released by the Security Innovation company, its holder.

[7] Another NTT-friendly option is $\phi(x) = x^n - x^{n/2} + 1$ with $n = 3 \cdot 2^s$. These settings are used in NPQCC/Falcon.

In the other direction, $f = \mathrm{NTT}^{-1}(\hat{f})$ and

$$f_i = \left(n^{-1}\gamma^{-i} \sum_{j=0}^{n-1} \hat{f}_j \omega^{-ij} \right) \bmod q.$$

Indeed,

$$\sum_{j=0}^{n-1} \hat{f}_j \omega^{-ij} = \sum_{j=0}^{n-1} \omega^{-ij} \sum_{k=0}^{n-1} \gamma^k f_k \omega^{kj} = \sum_{k=0}^{n-1} \gamma^k f_k \sum_{j=0}^{n-1} \omega^{(k-i)j} = n\gamma^i f_i \pmod q.$$

Here we use the fact that $\sum_{j=0}^{n-1} \omega^{kj}$ equals n if $k = 0$, and 0 otherwise.

Note that we slightly "spin" the expressions for coefficients $\hat{f}_j$ using powers of γ. It facilitates reduction modulo $x^n + 1$ during multiplication of polynomials $f, g \in \mathbb{Z}_q[x]/(x^n + 1)$.[8] The coefficients of the resulting polynomial $h = fg$ have the form ($\gamma^n \equiv -1 \pmod q$):

$$h_i = \sum_{0 \leqslant k,l \leqslant n-1:\ k+l \equiv i \bmod n} \gamma^{k+l-i} f_k g_l \pmod q.$$

They can be calculated as $\mathrm{NTT}^{-1}(\mathrm{NTT}(f) \circ \mathrm{NTT}(g))$, where $\circ$ is coefficient-wise (very fast) multiplication of polynomials. Indeed,

$$\begin{aligned}
\sum_{j=0}^{n-1} \gamma^{-i} n^{-1} \hat{f}_j \hat{g}_j \omega^{-ij} &= \sum_{j=0}^{n-1} \gamma^{-i} n^{-1} \omega^{-ij} \left(\sum_{k=0}^{n-1} \gamma^k f_k \omega^{kj} \right) \left(\sum_{l=0}^{n-1} \gamma^l g_l \omega^{lj} \right) \\
&= \sum_{k,l=0}^{n-1} n^{-1} \gamma^{k+l-i} f_k g_l \sum_{j=0}^{n-1} \omega^{j(k+l-i)} \\
&= \sum_{0 \leqslant k,l \leqslant n-1 : k+l \equiv i \bmod n} \gamma^{k+l-i} f_k g_l \pmod q.
\end{aligned}$$

The similar approach works also for polynomial division as well as for addition and subtraction. Moreover, NTT can be used to test invertibility: f is invertible in R_q if and only if all the coefficients $\hat{f}_j$ are nonzero.

The important point is that NTT can be calculated in $O(n \log n)$ operations in $\mathbb{Z}_q$. More precisely, we can find the vector $\hat{\mathbf{f}}$ of coefficients of $\hat{f}$ by the following algorithm (it can be written in many other ways):

1. For $i = 0, 1, \ldots, n-1$:
   (a) $t_i \leftarrow \gamma^{rev(i)} f_{rev(i)}$.
2. $\mathbf{t} \leftarrow (t_0, t_1, \ldots, t_{n-1})$;
3. For $i = 0, 1, \ldots, s-1$ and for $j = 0, 1, \ldots, n/2 - 1$:
   (a) $r \leftarrow j - (j \bmod 2^{s-1-i})$;
   (b) $\hat{f}_j \leftarrow (t_{2j} + t_{2j+1}\omega^r) \bmod q$;
   (c) $\hat{f}_{j+n/2} \leftarrow (t_{2j} - t_{2j+1}\omega^r) \bmod q$;

[8] Without this "spin" we obtain the reduction modulo $x^n - 1$ not $x^n + 1$.

(d) $\mathbf{t} \leftarrow \hat{\mathbf{f}}$.

4. Return $\hat{\mathbf{f}}$.

If $n = 2^s$, then $\phi(x)$ is the $2n$-th cyclotomic polynomial: $\phi(x) = \prod_{j=1,3,\ldots,2n-1}(x - \zeta^j)$, where $\zeta = e^{\pi i/n}$ is the primitive $2n$-th root of unity, the root of $\phi(x)$ in $\mathbb{C}$. The Galois group $\mathrm{Gal}(\mathbb{Q}(\zeta)/\mathbb{Q})$ consists of the mappings $\zeta \mapsto \zeta^j$, where j runs over $\mathbb{Z}^*_{2n}$. The group is thus isomorphic to $\mathbb{Z}^*_{2n}$ and contains n elements.

The group $\mathrm{Gal}(\mathbb{Q}(\zeta)/\mathbb{Q})$ is considered too small relatively to n by the NPQCC/NTRUPrime authors. They propose to use the third option: $\phi(x) = x^n - x - 1$. It is known that the chosen $\phi(x)$ is irreducible over $\mathbb{Q}$. So if α is some root of $\phi(x)$ in $\mathbb{C}$, then $\mathrm{Gal}(\mathbb{Q}(\alpha)/\mathbb{Q})$ tends to be very large reaching the symmetric (maximum) group S_n of $n!$ elements.

We do not fully understand the reasons for using in LBC the polynomial rings which induces large Galois groups. We think that it is mostly about fears around "*worrisome algebraic structures*" (the phrase from NPQCC/NTRUPrime documents), not about specific attacks. It is interesting that in 2016, the NTRU authors posted a draft that they had announced at Crypto 1996. In this draft, they also proposed to use $\phi(x) = x^n - x - 1$: "*This would slow computations somewhat, while providing greater mixing of the coeffcients.*" [9]

The NPQCC/NTRUPrime authors go further and impose additional restrictions on R_q:

1) modulus q is prime and $\phi(x) = x^n - x - 1$ is irreducible over $\mathbb{Z}_q$. So the ring R_q becomes a field of q^n elements. The modulus q which provides irreducibility is called "inert";

2) the dimension n is prime, so the field R_q is a prime-degree extension of the basic field $\mathbb{Z}_q$. This restriction is motivated by some approaches to cryptanalysis consisting in dispatching a target lattice problem between sublattices (see [95]).

The general approach to the choice of parameters in NPQCC/NTRUPrime is expressed as: "*prime-degree large-Galois-group inert-modulus lattice-based cryptography*".

The authors of NPQCC/NTRU-HRSS don't agree with this approach. They emphasize that "worrisome structures" are not related to real attacks. Instead of $\phi(x) = x^n - x - 1$, they propose to use $\phi(x) = x^{n-1} + \ldots + x + 1$. Here n is prime and $\phi(x)$ is, in fact, the classic NTRU polynomial $x^n - 1$ divided by $x - 1$.

If n is prime (again the classic NTRU case), then $\phi(x)$ is $\Phi_n(x)$, the n-th cyclotomic polynomial built by $\zeta = e^{2\pi i/n}$. The corresponding group $\mathrm{Gal}(\mathbb{Q}(\zeta)/\mathbb{Q})$ is isomorphic to $\mathbb{Z}^*_n$. The group is small that contradicts the recipe of the NPQCC/NTRUPrime team.

The main motivation for migrating to $\Phi_n(x) = (x^n - 1)/(x - 1)$ is to avoid the "dangerous" factor $x - 1$ in $x^n - 1$. This factor can potentially reveal information about secret polynomials of the ring R_q. The similar approach is used in NPQCC/LIMA, NPQCC/Round2 and its upgrade version NPQCC/Round5.

Additional restrictions in NPQCC/NTRU-HRSS are to make $\Phi_n(x)$ irreducible both modulo q and $p = 3$. We see the repetition of the "inert-modulus" idea.

The dimension n. The degree n of $\phi(x)$, which is also the dimension of the related lattices, determines the strength of these lattices. As we have seen in 5.2.3, the strength is increasing exponentially with n.

[9] See `https://web.securityinnovation.com/hubfs/files/ntru-orig.pdf`.

In LBC, n usually varies from ≈ 400 to ≈ 2000. For example, $443 \leqslant n \leqslant 2048$ for NPQCC candidates. The values $n \leqslant 500$ seem to be unsatisfactory small, the values $n \geqslant 1500$ seem to be unreasonably large.

The modulus q. In the classic NTRU, as well as in its front-end version NPQCC/NTRUEncrypt, the polynomial $\phi(x) = x^n - 1$ is used with $q = 2^s$. Such a modulus facilitates reduction on usual binary computers. If q has order $t \pmod n$, then $x^n - 1$ is factorized over $\mathbb{Z}_q$ into a product of $x-1$ and $(n-1)/t$ irreducible polynomials each of degree t. If t is small, then random polynomials of $\mathbb{Z}_q[x]$ would have a lot of common factors with $\phi(x)$ which can yield some attacks. To avoid such weakness, q is chosen in such way that t is large: preferably $n-1$ or $(n-1)/2$.

The option $q = 2^s$ is also used in NPQCC/Round2 and NPQCC/Round5 with $\phi(x) = (x^n-1)/(x-1)$. On the other hand, in NPQCC/LIMA and NPQCC/NTRU-HRSS polynomials $(x^n-1)/(x-1)$ are used with prime q. It is interesting that in NPQCC/LIMA it holds that $q \equiv 1 \pmod{2^e(n-1)}$ and n is a so-called safe prime: $n = 2p+1$, where p is another prime.

If $\phi(x) = x^n+1$ and n is a power of 2, then to launch NTT we must have an element $\gamma \in \mathbb{Z}_q$ of order $2n$. It is achievable if q is prime and $q \equiv 1 \pmod{2n}$. Indeed, in this case $\mathbb{Z}_q$ is a field, there exists an element $\alpha \in \mathbb{Z}_q$ of order $q-1$ and γ can be defined as $\alpha^{(q-1)/2n}$. Since γ has order $2n$, $\gamma^n \equiv -1 \pmod q$ and γ is a root of x^n+1 over $\mathbb{Z}_q$. All other roots also lay into $\mathbb{Z}_q$, so x^n+1 splits over $\mathbb{Z}_q$. This fact is indirectly used in NTT.

The settings "$n = 2^s$, q is prime, $q \equiv 1 \pmod{2n}$" are very popular among candidates of NPQCC. For the safe zone $500 \lessapprox n \lessapprox 1500$ there exist only two suitable options: $(n, q) = (512, 12289)$ and $(n, q) = (1024, 12289)$. Here $q = 12289$ is the smallest integer which satisfies the required conditions with $n = 512$ and $n = 1024$.

The parameters $(n, q) \in \{(512, 12289), (1024, 12289)\}$ are used in NPQCC/KCL, NPQCC/Falcon, NPQCC/HILA5, NPQCC/NewHope.

Other suitable primes $q = 7681$ $(n = 256)$, $q = 120883$ $(n = 512, 1024)$ and $q = 40961$ $(n = 512)$, $q = 8380417$ $(n = 256)$ are used in NPQCC/CRYSTALS-KYBER, NPQCC/Ding, NPQCC/EMBLEM and NPQCC/CRYSTALS-Dilithium respectively. Since $n = 256$ is too small for secure LBC, the target lattices are build from k-dimensional *modules* over R_q, $k \geqslant 2$. Such "dimension stretching" is often used in LWE cryptography (see MLWE in 5.2.7).

In NPQCC/qTESLA three gigantic (but also suitable) primes are used: $q = 8058881$ $(n = 1024)$, $q = 12681217$ $(n = 2048)$ and $q = 27627521$ $(n = 2048)$. These primes have the form $2^r - 2^s + 1$ to enable efficient modular reduction on constrained (8-bit and 16-bit) platforms.

Some NPQCC submissions (NPQCC/KINDI, NPQCC/Lizard, NPQCC/Saber) use $q = 2^s$. Although these moduli exclude the use of NTT, the authors of submissions provide reasons for such exclusion. The main reason is that the reduction modulo 2^s is almost for free.

In NPQCC/LAC $q = 251$ is used with $n = 512, 1024$. The authors write: "*We cannot use NTT directly in LAC. Fortunately, when* $q < 256$*, we can gain a speed improvement for the polynomial multiplications.*"

Additonal details on the parameters of NPQCC submissions can be found in Table 5.1.

5.2.6 NTRU Encryption

NTRU cryptography. The NTRU cryptosystem was invented in 1996 by J. Hoffstein,

J. Piher, and J. Silverman.[10] There exist several variants of the basic scheme proposed by the same team, all these variants are usually attributed as *NTRU Classic*. NTRU Classic can be considered as an *area* in the NTRU *branch* of the LBC *platform*. The last representative of NTRU Classic is the NPQCC/NTRUEncrypt submission. The submitters are C. Chen, J. Hoffstein, W. Whyte, and Z. Zhang.

The first version of NTRU uses the polynomial $\phi(x) = x^n - 1$ with a capital N instead of n. The corresponding ring $R = \mathbb{Z}[x]/(x^N - 1)$ was called by authors "*Nth degree Truncated polynomial Ring Units*", hence the abbreviation NTRU. In the latest versions of NTRU other options for $\phi(x)$ are allowed, for example, in NPQCC/NTRUEncrypt the polynomial $\phi(x) = x^{1024} + 1$ is used with $q = 2^{30} + 2^{13} + 1$.

The NTRU branch is widely represented in NPQCC. There are 3 PKE/KEM submissions: NTRUEncrypt, NTRUPrime, NTRU-HRSS, and 2 DS submissions: Falcon, pqNTRUSign. Moreover, the basic idea of NTRU influences other submissions, even ones from other platforms. For example, NPQCC/LAKE and NPQCC/LOCKER use this idea.

Further we use the centered representation of $\mathbb{Z}_q$ (see 5.2.4), that is, identify $\mathbb{Z}_q$ with the set $\mathbb{Z} \cap [-q/2, q/2)$.

NTRU lattices. So what is the idea? Let f and g be random small polynomials from $R_q = \mathbb{Z}_q[x]/(\phi(x))$ and let f be invertible in R. Alice keeps (f, g) in secret, this pair is her private key. Alice calculates $h = g/f$ and publishes it as her public key. The relation between keys allows Alice to setup PKE, KEM, DS and other cryptographic mechanisms.

Usually coefficients of f and g are from $\mathsf{T} = \{\pm 1, 0\}$. Thus, the private key is very compact: it can be represented by $2n$ trits. The public key h is also compact: it needs approximately $n \log_3 q$ trits. In LBC, the reasonable safe zones are $500 \lessapprox n \lessapprox 1500$ and $2^{10} \lessapprox q \lessapprox 2^{16}$. So we can expect public keys of approximately $3000 \div 15000$ trits. It is much less than the usual amount of key material used in CBC or MBC.

Before considering how to launch PKE/KEM, let us discuss how hard it is to reconstruct (f, g) from h, that is, to find a private key from a public one. Interestingly, the NTRU problem $h = g/f \mapsto \{g, f\}$ is very similar to the famous Integer Factorization Problem $n = pq \mapsto \{p, q\}$ (the RSA form), where p and q are disctinct primes.

First of all, if f and g are generated with enough entropy, then h tends to be a random polynomial of R_q and, therefore, tends to have rather large (in magnitude) coefficients.

For example, in [175] it was proved that coefficients of h are uniformly distributed over $\mathbb{Z}_q$ given that coefficients of f and g have a discrete Gaussian distribution over $\mathbb{Z}$ with zero mean and variance of order $n\sqrt{q \log(nq)}$ provided that NTT-friendly parameters are used. The authors of [175] use this result to push NTRU in the Provable Security context.

Let H be a matrix that represents h (see 5.2.5) and let

$$B = \begin{pmatrix} I_n & H \\ 0 & qI_n \end{pmatrix}.$$

The matrix B generates an NTRU lattice $\mathcal{L} = \mathcal{L}(B)$ of dimension $2n$. The matrix consists of rather large basis vectors. Indeed, the first n rows of B contain random coefficients of h, each of the next n rows contains a rather large entry q.

[10] The first paper on NTRU was submitted to CRYPTO'97 and was rejected! A great surprise given the impact NTRU, the "grandfather of LBC encryption schemes" [121], had.

Since $h = g/f$, there exist a polynomial t such that $fh = (g + qt) \mod \phi$. It could be rewritten in the matrix form: $FH = G + qT$. If we (left) multiply B by $(F \ -T)$, we will obtain the matrix $(F\ G)$. In other words, $\mathcal{L}$ contains the rather short vector $(\mathbf{f}, \mathbf{g})$, its norm is of order $\sqrt{n}$.[11]

Thus, a public polynomial h generates a lattice which short vector is derived from secret polynomials f and g. We are in the context of the classical SVP or Approximate-SVP problems (see 5.2.2).

Let us suppose that both f and g contain approximately third zeros, and their nonzero coordinates are ± 1. It is indeed the case in NTRU Classic. Since $\det B = q^n$, the lattice $\mathcal{L}$ tends to have vectors of minimum length

$$\lambda_1(\mathcal{L}) \approx \sqrt{\frac{nq}{\pi e}}.$$

But we always know that $\mathcal{L}$ contains the vector $(\mathbf{f}, \mathbf{g})$ of length $\approx \sqrt{4n/3}$. It is natural to introduce the ratio of the known shortest length to the expected shortest length as if $\mathcal{L}$ will be a random lattice. This ratio is approximately

$$\frac{\sqrt{4n/3}}{\sqrt{nq/(\pi e)}} = \sqrt{\frac{4\pi e}{3q}}.$$

We see that the shortest vectors of NTRU lattices are unusually short. There is a high probability that the shortest vectors are $(\mathbf{f}, \mathbf{g})$ and its rotations.

It seems that the shorter vector, the harder to find it. But it is not entirely true. There exist so-called combinatorial attacks that seem to work best when the secret is small. We will discuss them in Section 5.7.

It is quite natural to rewrite B as

$$\begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix}$$

implicitly implying matrices in the places of polynomials. Such notations are very common in LBC. The corresponding lattice $\mathcal{L}(B)$ can be written as $R^2 \begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix}$ and, therefore, stands as a 2-dimensional module over $\mathbb{Z}[x]/(\phi(x))$.

The important point regarding NTRU lattices is that they are not ideal lattices. An ideal lattice is generated by an ideal $I \subset R$ which polynomials f are represented by vectors $\mathbf{f}$. An ideal lattice has several weaknesses. One of them is that if I is principal, that is, $I = fR$ for some $f \in R$, then the shortest vector of the corresponding ideal lattice can be found in sub-exponential or even quantum polynomial time [66]. Fortunately, NTRU lattices are not ideal, because ideal lattices are 1-dimensional over R.

Encryption. Let us move on to the encryption. Now we intensively interpret f, g, h and other polynomials of R_q as polynomials of R. These polynomials are implicitly reduced modulo ϕ and can be additionally reduced $\mod s\, q$ or $\mod s\, p$. Here p is a small positive integer coprime to q (usually $p = 3$). The double reduction $\mod \phi \mod s\, q$ brings polynomials to the

[11] The matrix $(F\ G)$ is only a half of a short basis of $\mathcal{L}$. Interestingly, in NPQCC/Falcon it is proposed to use the second half. It has the form $(F'\ G')$, where the matrices F', G' correspond to polynomials f', g' such that $fg' - f'g = q \pmod \phi$.

ring R_q. Let f be invertible not only in R_q, that is, $\bmod \phi \bmods q$ but also in R_p, that is, $\bmod \phi \bmods p$.

To encrypt a small ($2\|m\|_\infty \leqslant p$) polynomial m, Bob chooses a small random polynomial r and calculates the ciphertext

$$c \leftarrow (phr + m) \bmod \phi \bmods q.$$

To decrypt c, Alice calculates $t \leftarrow cf \bmod \phi \bmods q$. This polynomial can be rewritten as

$$(pgr + mf) \bmod \phi \bmods q.$$

The internal polynomial $pgr + mf$ is small because all its terms are short. Moreover, under some conditions $pgr + mf$ is *enough* small and reduction $\bmods q$ is not required:

$$t = (pgr + mf) \bmod \phi.$$

Now

$$tf^{-1} \bmod \phi \bmods p = m$$

and decryption is completed.

The main (and really clever) idea here is to suppress reduction $\bmods q$ by achieving small $pgr+mf$. The evidence that the idea is indeed smart is the fact that no significant improvements in the NTRU encryption are known over the past 20 years.

But, of course, improvements exist. Four of them are discussed below.

1. Mitigation against $m(1)$ leakage. Classic NTRU with $\phi(x) = x^n - 1$ can leak information about m if $r(1)$ is known (it is indeed the case for NTRU Classic). Indeed, since $(x-1) \mid (x^n-1)$, we can deduce from $c(x) \equiv ph(x)r(x) + m(x) \bmod (x^n - 1) \bmods q$ the congruence

$$c(x) \equiv ph(x)r(x) + m(x) \bmod (x - 1) \bmods q.$$

In fact, we decrease the dimension from n to 1 saving the encryption equation! In particular,

$$c(1) \equiv ph(1)r(1) + m(1) \bmod q$$

and $c(1)$ reveals information about $m(1)$.

Note that $m(1)$ leaks even if $r(1)$ isn't exactly known. An adversary finds small $f(1)$, $g(1)$ such that $h(1)f(1) \equiv g(1) \bmod q$. After that the adversary reveals $m(1)$ as if she works with NTRU of dimenision 1 knowing the private key $(f(1), g(1))$.

To overcome the weakness, current NTRU Classic specifications prohibit m that have very unbalanced numbers of 0's and ± 1's so that $m(1)$ doesn't actually contain useful information about m.

NPQCC/NTRU-HRSS proposes another approach: avoid a "dangeorus" factor $x-1$ in $x^n - 1$ by moving to $\phi(x) = (x^n - 1)/(x - 1)$. In fact, in NPQCC/NTRU-HRSS the polynomials f, g, r, m are taken from $\mathbb{Z}_q[x]/(\phi(x))$ and they are "lifted" to $\mathbb{Z}_q[x]/(x^n - 1)$ using a special technique.

2. Rounded NTRU. In NPQCC/NTRUPrime, m is not transmitted but generated during encryption, that is, the KEM settings are used. A ciphertext c is determined by rounding each coefficient of hr to the nearest multiple of 3. Such rounding means an implicit addition of a polynomial m which coefficients are from $\{\pm 1, 0\}$. The NPQCC/NTRUPrime authors explain

2 advantages of their "Rounded NTRU": first, it reduces the space required to transmit $m+hr$; second, the fact that m is determined by r simplifies protection against chosen-ciphertext attacks.

3. pNE (Provable secure NTRU Encryption). To prove IND-CPA security of NTRU encryption, it is necessary to show that the pair (h, phr) is indistinguishable from a pair of random elements of R_q. But (h, phr) is easily distinguishable. Indeed, $h = g/f$ doesn't have the uniform distribution over R_q (we already discussed this fact). Moreover, dividing phr by ph, we get the unnaturally small for a random pair polynomial r.

To overcome these Provable Security issues, authors of [175] propose: 1) use large q and generate f, g using a discrete Gaussian distribution with a large variance; 2) determine a ciphertext as $p(hr + e) + m$, where e is an additional random small polynomial which stands as noise.

Although the pNE proposal was generally regarded as impractical,[12] it was included in the NPQCC/NTRUEncrypt. We think that it is a very wise decision: the third set of long-term parameters (a *paramset*) is for ProvSec addicts, the first two paramsets are for adepts of common sense. In any case, results of [175] are very important because they illustrate that the general design of NTRU encryption is sound.

4. Without inversions mods p. Another modification of NTRU is to change $h = g/f$ with $h = g/(1 + pf)$. Now we don't need to inverse $f \bmod s\, p$ and can determine m as follows:

$$m \leftarrow c(1 + pf) \bmod \phi \bmod s\, q \bmod s\, p.$$

Decryption works if the polynomial $pgr + pmf + m$, which is slightly greater than the previous one, is small. This modification seems very natural, it is used in NPQCC/NTRUEncrypt. Let us concentrate further on NTRU with this modification.

Moreover, let us further consider only the case $p = 3$ which is the most suitable for our ternary logic. This case prevails in NTRU encryption with rare exceptions. These exceptions are, for example, the 3rd (ProvSec) paramset of NPQCC/NTRUEncrypt or a scheme from [70]. In both cases $p = 2$.

NTRU Problem. In the settings made, an adversary who attacks NTRU has to solve the following problem.

Problem (NTRU Problem)**.** Given (ϕ, n, q), probabilistic distibutions χ_f, χ_g, χ_r over $R_3 = \mathbb{Z}_3[x]/(\phi(x))$ and polynomials $h = g/(1 + 3f)$, $3hr + m$ built from

$$f \xleftarrow{\chi_f} R_3, \quad f \xleftarrow{\chi_g} R_3, \quad r \xleftarrow{\chi_r} R_3, \quad m \in R_3,$$

find m. The distribution χ_f used should block non-invertible polynomials $1 + 3f$.

Under KEM settings, NTRU Problem is slightly modified allowing m to also be randomly generated according to an additional distribution χ_m. Usually, $\chi_r = \chi_m$.

In NTRU Classic, distributions χ_f, χ_g, χ_r produce ternary polynomials with strictly predefined numbers of 0's and ± 1's.

In the latest instantiations of NTRU Encryption, the distributions tend to be not so restrictive. For example, χ_r in NPQCC/NTRU-HRSS and χ_g in NPQCC/NTRUPrime are uniform over the set of ternary polynomials $\bmod \phi$.

[12] See, for example, `https://www.youtube.com/watch?v=1Sybp6-cbTA&feature=youtu.be`.

In [70], the distributions are expanded to the overall R_q and these distributions are discrete Gaussian. It makes h more uniform and to decrease decryption errors. So [70] follows the pNE approach proposed in [175]. This approach is also used in NPQCC/NTRUEncrypt with the third (ProvSec) set of long-term parameters.

Further information on random data generation in LBC is provided in 5.2.8.

Generally speaking, to solve NTRU Problem there is no need to find the short vector $(\mathbf{f}, \mathbf{g})$. It is sufficient to find a vector of the NTRU lattice which is the closest to $(\mathbf{0}, \mathbf{c})$. Indeed, from $c \equiv 3hr + m \bmod q$ it follows that there exist a polynomial $t \in R$ such that

$$c = 3hr + m - qt$$

or

$$(\mathbf{0}, \mathbf{c}) = \mathbf{v} + (-\mathbf{r}, \mathbf{m}), \quad \mathbf{v} = (\mathbf{r}, -\mathbf{t}) \begin{pmatrix} 1 & 3h \\ 0 & q \end{pmatrix},$$

in matrix form. Here $(-\mathbf{r}, \mathbf{m})$ is the difference between the given vector $(\mathbf{0}, \mathbf{c})$ and the vector $\mathbf{v}$ of the NTRU lattice $R^2 \begin{pmatrix} 1 & 3h \\ 0 & q \end{pmatrix}$. Since the distance $\|(\mathbf{r}, \mathbf{m})\|$ is very short (it doesn't exceed $\sqrt{2n}$), $\mathbf{v}$ is most likely a solution of CVP$(\mathbf{0}, \mathbf{c})$.

In [70], NTRU Problem is alternatively formulated as SVP. Authors introduce the q-ary lattice

$$\Lambda = \{(\mathbf{u}, \mathbf{v}, w) \colon u, v \in R,\ w \in \mathbb{Z}, 3hu + v - zc \equiv 0 \bmod q\}.$$

It is an additive subgroup of $\mathbb{Z}^{2n+1}$ and thus a lattice. The lattice contains the short vector $(\mathbf{r}, \mathbf{m}, 1)$ which seems to be a solution of SVP.

Decryption failures. Let us recall that to correctly decrypt data, the polynomial $3gr + 3mf + m$ must be small. More precisely, each its coefficient must be less than $q/2$:

$$\|3gr + 3mf + m\|_\infty < q/2.$$

We can easily satisfy this condition by increasing q. But the larger q, the slower algorithms and the greater sizes of public keys and ciphertexts. Another problem is that security generally degrades as q grows with fixed n. For example, there exist the so-called subfield attack which finds f, g by h in polynomial time provided that $q = 2^{\Omega(\sqrt{n \log \log n})}$ (see [129] for the latest details).

Historically, in NTRU Classic q is not too enough large to eliminate decryption failures: they can occur with controllable small probabilities. For example, in NPQCC/NTRUEncrypt, these probabilites for 3 standard paramsets don't exceed 2^{-196}, 2^{-112} and 2^{-80}. For comparison, the declared strengths (classical, not quantum) of these paramsets are 2^{85}, 2^{159}, 2^{198}. Interestingly, probabilities of decryption failures and declared strengths are weakly correlated.

Decryption failures seem to be a reasonable design solution if it increases the effectiveness of algorithms and compactness of their data. NPQCC allows probabilities of decryption failures below 2^{-64}. On the other hand, decryption failures are strongly undesirable in many situations. Entities of MAM2, who send data to each other, should have enough strong guarantees that their keyloads will be decrypted. Of course, it seems that the decryption failure rates should correlate with the strengths of underlying lattices.

Although NPQCC/NTRUEncrypt is quite liberal to decryption failures, other submissions to NPQCC are not. Both NPQCC/NTRUPrime and NPQCC/NTRU-HRSS completely eliminate decryption failures. The NPQCC/NTRUPrime authors motivate eliminating of decryption

failures: "*We prefer to guarantee that decryption works, making the security analysis simpler and more robust.*" (It means that decryption should work not only in practice, it also should follow standard ProvSec models).

NAEP. Additional weaknesses of NTRU encryption are the following.

Repeated messages. Let Bob encrypt the same message m twice. An adversary intersects $c = 3hr + m$, $c' = 3hr' + m$ and finds

$$(r' - r) = (c - c')/(3h)^{-1}.$$

Since r, r' are small (they are ternary), the adversary can reveal a lot of their coordinates r_i, r'_i knowning the differences $r_i - r'_i$.[13]

Malicious ciphertexts. Under CCA settings, an adversary can send to Alice arbitrary $c \in R_q$ and obtain $c(1 + 3f) \bmod q \bmod 3$. The adversary sends h and obtains g, one half of a private key.

These examples show that the overall encryption flow must be strengthened by means of additional cryptographic mechanisms irrelevant to lattice issues.

In NTRU Classic these mechanisms are called NAEP (NTRU Asymmetric Encryption Padding). The first version of NAEP works as follows:

1. Bob processes a ternary message m which length is $n - l < n$. Bob generates a random $b \in \mathbf{T}^l$ and calculates $r \leftarrow G(m \parallel b)$. Here G is a hash function that maps from $\mathbf{T}^n$ to R_3.

2. Bob calculates $t \leftarrow 3hr$ and uses another hash function H which takes t and returns ternary string of length n.

3. Bob determines ternary $m' \leftarrow ((m \parallel b) + H(t)) \bmod 3$. If m' doesn't satisfy security conditions (numbers of 0's and ± 1's are unbalanced), then Bob goes to Step 1.

4. Bob determines a ciphertext $c = t + m'$. As usual, ternary words are identified with ternary polynomials.

5. Alice finds m' using standard NTRU calculation and then determines $t = c - m'$.

6. Alice calculates $H(t)$ and $(m \parallel b) \leftarrow (m' - H(t)) \bmod 3$.

7. Alice determines $r \leftarrow G(m \parallel b)$ and test if $3hr = t$.

8. Alice returns m if all checks are successful and $\perp$ otherwise.

We see that NAEP indeed protects against mentioned attacks: it randomizes input data, it blocks answers on invalid decrypt queries. Internally, all random coins used during encryption (b, r) can be reconstructed from ciphertext. This is the main technique that allows proving IND-CCA security of NTRU encryption (see [116]). We see also that NAEP is rather heavy: a lot of cryptographic calculations, Alice not only decrypts but also repeats Bob's encryption.

The latest NAEP version, used in NPQCC/NTRUEncrypt, is slightly lighter:

[13] An even more serious problem is when Bob uses r twice. In this case an adversary finds $m' - m = c - c'$ and reconstructs coordinates of m, m'.

1. Bob processes a binary message m which length doesn't exceed $n - 173$. The message is aligned up to length n with pseudorandom trits. The resulting message is still denoted m. First its bits are interpreted as trits.

2. Bob calculates $r \leftarrow G(m \parallel h)$. Note that now the public key h directly affects the encryption result.

3. Bob calculates $t \leftarrow 3hr$ and $m' \leftarrow (m - H(t)) \bmod 3$. Here H maps from R_q to R_3.

4. Bob determines a ciphertext $c = t + m'$.

5. Alice finds m' using standard NTRU calculation and then determines $t = c - m'$.

6. Alice calculates $H(t)$ and $m \leftarrow (m' + H(t)) \bmod 3$.

7. Alice finds $r \leftarrow G(m)$ and test if $3rh = t$.

8. Alice returns m if all checks are successful and $\perp$ otherwise.

In NPQCC/NTRUPrime, NPQCC/NTRU-HRSS much more easier cryptographic padding is used. These submissions are of KEM type, so it is no need for such strong security guarantees as IND-CCA. For example, in NPQCC/NTRUPrime, an encapsulated key K and a confirmation C are produced by hashing of r. The ciphertext c (in fact, rounded r) is accompanied with C. We see that the length of the ciphertext increases and this is not very good. During decryption the confirmation C is checked.

5.2.7 LWE Encryption

Basic LWE schemes. In his pioneering paper [167], O. Regev accompanied the introduced LWE problem with a PKE scheme. Let us start with the simplest variant of the scheme that allows encrypting only single-trit messages.

A private key is a vector $\mathbf{s} \xleftarrow{\chi_s} \mathbb{Z}_q^n$ which is generated in accordance with a *secret distribution*[14] χ_s. A public key consists of a matrix $A \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $m \geqslant n$, and a vector $\mathbf{b} \leftarrow \mathbf{s}A + \mathbf{e}$. Here $\mathbf{e} \xleftarrow{\chi} \mathbb{Z}_q^m$. The *error distribution* χ tends to produce short vectors $\mathbf{e}$ that stand as errors. We can even write $\mathbf{b} \approx \mathbf{s}A$.

As originally proposed in [167], χ is the rounded Gaussian distribution with a small variance. However, nowadays LWE cryptography usually uses the discrete Gaussian distribution. We do not pay attention to such subtleties and present both distributions as Gaussian (normal).

Initially, the uniform distribution was used as χ_s. It was later shown that other options, for example, $\chi_s = \chi$, are also secure. A lot of LWE submissions to NPQCC use secret distributions that tend to produce short vectors. In Table 5.1 we mention the following ones:

- $R[-a, a]$ — coordinates of $\mathbf{s}$ are taken at random from the interval $[-a, a] \subset \mathbb{Z}_q$ where a is small;

- $R_d[-1, 1]$ — coordinates of $\mathbf{s}$ are taken at random from $[-1, 1]$ and exactly d coordinates are nonzero.

[14] In the sense of "a distribution of secrets".

It is easy to generate private and public keys $sk = \mathbf{s}$ and $pk = (A, \mathbf{b} \approx \mathbf{s}A)$. But to determine sk from pk, it is necessary to solve the hard (with properly configured parameters) Search-LWE problem. So we are in the usual cryptographic paradigm "hard-to-invert".

A trit μ, which Bob wants to encrypt, is represented as $f(\mu) = \mu \lfloor q/3 \rceil \in \mathbb{Z}_q$. A "quasi-inverse" f^{-1} maps $\alpha \in \mathbb{Z}_q$ to

$$f^{-1}(\alpha) = \lfloor \alpha/(q/3) \rceil .$$

When encrypting, Bob chooses a vector $\mathbf{r} \xleftarrow{R} \mathbf{T}^m$ that is interpreted as an element of $\mathbb{Z}_q^m$. A ciphertext is constructed as a pair $c = (\mathbf{u}, v)$ in which

$$\mathbf{u} \leftarrow \mathbf{r}A^T, \quad v \leftarrow \mathbf{r}\mathbf{b}^T + f(\mu).$$

Alice decrypts c as follows:

$$\hat{\mu} \leftarrow f^{-1}(v - \mathbf{u}\mathbf{s}^T) = f^{-1}(\mathbf{r}\mathbf{e}^T + f(\mu)).$$

Decryption works (that is, $\hat{\mu} = \mu$) if $\mathbf{r}\mathbf{e}^T$ is less than $q/6$. Adjusting the parameters (n, q, m, χ), we can achieve that decryption failures occur with a reasonable small probabilities. One of the settings is $m \approx (n+1)\log_3 q$. It provides a good balance between efficiency and security (see [150, 159] for details).

Longer plaintexts. The basic scheme can be generalized: instead of encrypting a single trit μ, we can encrypt a vector $\mathbf{m} \in \mathbf{T}^l$. Now $\mathbf{s}$, $\mathbf{e}$ and $\mathbf{b}$ become matrices $S \xleftarrow{R} \mathbb{Z}_q^{l \times n}$, $E \xleftarrow{\chi} \mathbb{Z}_q^{l \times m}$ and $B \leftarrow SA + E$ respectively. Again $B \approx SA$. The action of f, f^{-1} is transferred onto vectors in a natural way. Encryption:

$$\mathbf{r} \xleftarrow{R} \mathbf{T}^m, \quad \mathbf{u} \leftarrow \mathbf{r}A^T, \quad \mathbf{v} \leftarrow \mathbf{r}B^T + f(\mathbf{m}).$$

Decryption:

$$\hat{\mathbf{m}} \leftarrow f^{-1}(\mathbf{v} - \mathbf{u}S^T) = f^{-1}(\mathbf{r}E^T + f(\mathbf{m})).$$

The new balance condition: $m \approx (n + l)\log_3 q$.

Normal forms. A public matrix A generates the lattice $\Lambda_q(A)$ to which belong rows of SA. The basis A of the lattice can be replaced with its HNF (Hermite Normal Form) $(I_n \; A')$ in which $A' \in \mathbb{Z}_q^{n \times (m-n)}$. We can store $n(m-n)$ entries of A' instead of nm entries of A and use $(I_n \; A')$ in place of A in the above calculations. The vector $\mathbf{b}$, the second part of the public key, is compressed up to $\mathbf{b}' \in \mathbb{Z}_q^{m-n}$ and used instead of $\mathbf{b}$ (see [150, 159] for details). In the new settings, χ_s usually coincides with χ.

Square matrices. In [141] it is proposed to use a square public matrix $A \in \mathbb{Z}_q^{n \times n}$ instead of an oblong rectangular one. The simplification of A is accompanied with the choice $\chi_s = \chi$. To maintain security, the second parts of ciphetexts are spoiled with additional errors: $c = (\mathbf{u}, v)$ where $\mathbf{u} = \mathbf{r}A$ and $v \overset{\chi}{\approx} \mathbf{r}\mathbf{b}^T + f(\mu)$. Alice decrypts c as usual. Decryption works since

$$v - \mathbf{u}\mathbf{s}^T \approx \mathbf{r}\mathbf{b}^T + f(\mu) - \mathbf{r}A\mathbf{s}^T = \mathbf{r}\mathbf{e}^T + f(\mu) \approx f(\mu).$$

p-valued plaintexts. Another generalization is to use p-valued vectors $\mu \in \mathbb{Z}_p$ instead of ternary ones. The implementation is easy: we need to replace $q/3$ in the above equations with q/p. Of course, we should use $p \ll q$.

The basic LWE schemes are presented by several submissions to NPQCC (see Table 5.1). Although these submissions use additional optimizations, the lengths of public keys and ciphertexts are still very large.

Table 5.1: LWE/LWR/RLWE/RLWR/MLWE/MLWR submissions to NPQCC

Submission	Primitive	Algebraic structure	χ_s	Failures
LWE ($\mathbb{Z}_q^{n\times m}$)				
EMBLEM	KEM, PKE	$\mathbb{Z}_{16777216}^{611\times 832}$ or $\mathbb{Z}_{16777216}^{770\times 1033}$	$[-1,1]$ or $R[-2,2]$	2^{-140}
Frodo	KEM, PKE	$\mathbb{Z}_{32768}^{640\times 640}$ or $\mathbb{Z}_{65536}^{976\times 976}$	normal	2^{-148} or 2^{-199}
LOTUS	KEM, PKE	$m = n = 576, 704, 832$, $q = 8192$	normal	2^{-256}
LWE / LWR ($\mathbb{Z}_q^{n\times m}$)				
Lizard	KEM, PKE	$n = 536, 663, \ldots, 1300$ $q = 1024, 2048, 4096$ $m = 1024, 2048$	$R_d[-1,1]$	$2^{-153\div 381}$
uRound2	KEM, PKE	$m = n = 500, 585, \ldots, 835$ $q = 32768$	$R_d[-1,1]$	$2^{-69\div 101}$
RLWE ($\mathbb{Z}_q[x]/(\phi(x))$)				
Ding	KEM	$\mathbb{Z}_{120883}[x]/(x^{512}+1)$ $\mathbb{Z}_{120883}[x]/(x^{1024}+1)$	normal	2^{-60}
KCL	KEM	$\mathbb{Z}_{12289}[x]/(x^{1024}+1)$	normal	$2^{-38\div 41}$
R.EMBLEM	KEM, PKE KEM, PKE	$\mathbb{Z}_{16384}[x]/(x^{512}+1)$ $\mathbb{Z}_{65536}[x]/(x^{512}+1)$	$[-1,1]$ $[-1,1]$	2^{-140}
KCL	KEM	$\mathbb{Z}_{12289}[x]/(x^{1024}+1)$	normal	
LAC	KEM, PKE	$\mathbb{Z}_{251}[x]/(x^n+1)$ $n = 512, 1024$	normal	$2^{-115\div 239}$
LIMA-2p	KEM, PKE	$\mathbb{Z}_{133121}[x]/(x^{1024}+1)$ $\mathbb{Z}_{184321}[x]/(x^{2048}+1)$	normal	0
LIMA-sp	KEM, PKE	$\mathbb{Z}_q[x]/((x^m-1)/(x-1))$ $m = 1019, 1307, 1823, 2063$ q – large primes $(24-25$ bits$)$	normal	0
NewHope	KEM, PKE	$\mathbb{Z}_{12289}[x]/(x^{512}+1)$ $\mathbb{Z}_{12289}[x]/(x^{1024}+1)$	normal	2^{-213} 2^{-216}
RLWE / RLWR ($\mathbb{Z}_q[x]/(\phi(x))$)				
R.Lizard	KEM, PKE	$n = 1024, 2048$ $q = 1024, 2048, 4096$	$R_d[-1,1]$	$2^{-153\div 381}$
nRound2	KEM, PKE	$\mathbb{Z}_q[x]/((x^m-1)/(x-1))$ $q = 1319, 1949, \ldots, 3343$ $m = 401, 443, \ldots, 709$	$R_d[-1,1]$	$2^{-164\div 245}$
uRound2	KEM, PKE	$\mathbb{Z}_q[x]/((x^m-1)/(x-1))$ $q = 4096, 16384, 32768$ $m = 419, 421, , \ldots, 709$	$R_d[-1,1]$	$2^{-137\div 300}$
MLWE ($(\mathbb{Z}_q[x]/(\phi(x)))^k$)				
KCL	KEM	$(\mathbb{Z}_{7681}[x]/(x^{256}+1))^3$	normal	$2^{-36\div 106}$
KINDI	KEM, PKE	$(\mathbb{Z}_q[x]/(x^{256}+1))^k$ $k = 2, 3, 5$, $q = 8192, 16384$	$R[-a,a]$ $a = 2, 4$	$2^{-165\div 276}$
Kyber	KEM, PKE	$(\mathbb{Z}_{7681}[x]/(x^{256}+1))^k$, $k = 2, 3, 4$	normal	2^{-140}
MLWR ($(\mathbb{Z}_q[x]/(\phi(x)))^k$)				
Saber	KEM, PKE	$(\mathbb{Z}_{8192}[x]/(x^{256}+1))^k$, $k = 2, 3, 4$	normal	$2^{-36\div 106}$

Several words about Table 5.1.[15] It shows that LWE cryptography is "on the march". In fact, this is the most numerous unit of not only the LBC platform but also all Post-Quantum crypto in NPQCC. LWE cryptography has been dynamically growing in the last 10 years, new breakthrough appears almost every year. Suffice it to say that the table does not present submissions that are based on the novel ILWE and PLWE problems. We will briefly discuss these problems and submissions separately.

The last column of the table presents upper bounds on the probabilities of decryption failures. To control these probabilities, we usually should use rather large q. Interestingly, in NPQCC/LAC it is achieved using BCH codes with the comparatively small $q = 251$.

LWR schemes. The LWR (Learning with Rounding) problem was introduced in [19]. Let $p < q$ and $\mathbb{Z}_q$ be divided into p contiguous non-overlapping intervals of roughly q/p elements each. A *rounding function* $\lfloor\cdot\rceil_p$ assigns to $x \in \mathbb{Z}_q$ the index of the interval to which x belongs. For example, $\lfloor x\rceil_p = \lfloor x(p/q)\rceil \bmod p$, where x is interpreted as a real number.

Problem (Decision-LWR)**.** Let O_0 be an oracle which is instantitated with $\mathbf{s} \in \mathbb{Z}_q^n$ and outputs the pairs

$$(\mathbf{a}, \lfloor \mathbf{s}\mathbf{a}^T \rceil_p), \quad \mathbf{a} \xleftarrow{R} \mathbb{Z}_q^n.$$

Let O_1 be another oracle which outputs the pairs

$$(\mathbf{a}, \lfloor u \rceil_p), \quad (\mathbf{a}, u) \xleftarrow{R} \mathbb{Z}_q^{n+1}.$$

Let $d \xleftarrow{R} \{0, 1\}$. Given n, q, χ and an access to O_d, find d.

In [19] it is shown that if errors $e \xleftarrow{\chi} \mathbb{Z}_q$ are small and the ratio q/p is large, then $\lfloor \mathbf{a}\mathbf{s}^T + e \rceil_p = \lfloor \mathbf{a}\mathbf{s}^T \rceil_p$ with an overwhelming probability. Therefore, the pairs $(\mathbf{a}, \lfloor \mathbf{a}\mathbf{s}^T \rceil_p)$ are hardly to distinguish from the pairs $(\mathbf{a}, \lfloor \mathbf{a}\mathbf{s}^T + e \rceil_p)$. But due to the hardness of Decision-LWE, the latter pairs are hardly to distinguish from $(\mathbf{a}, \lfloor u \rceil_p)$ with $u \xleftarrow{R} \mathbb{Z}_q$. In result, Decision-LWR is hard (as well as Search-LWR).

LWR can be used instead of LWE to build PKE or KEM algorithms. Let us describe how LRW encryption is instantiated in NPQCC/Lizard.

Alice generates a usual private $\mathbf{s}$ and public $(A, \mathbf{b} = \mathbf{s}A + \mathbf{e})$ keys. To encrypt a trit μ, Bob Bob generates $\mathbf{r} \xleftarrow{R} \mathbf{T}^n$, calculates

$$\mathbf{u} \leftarrow \lfloor (3/q)(\mathbf{r}A^T \bmod q) \rceil \bmod p, \quad v \leftarrow \lfloor (p/3)\mu + (p/q)(\mathbf{r}\mathbf{b}^T \bmod q) \rceil \bmod p,$$

the parts of a ciphertext $c = (\mathbf{u}, v)$. Alice decrypts c as follows:

$$\hat{\mu} \leftarrow \left\lfloor \frac{3}{p}(v - \mathbf{u}\mathbf{s}^T) \right\rceil \bmod 3.$$

Decryption works since

$$\frac{3}{p}(v - \mathbf{u}\mathbf{s}^T) \approx \frac{3}{p}\Big((p/3)\mu + (p/q)\mathbf{r}A^T\mathbf{s}^T + (p/q)\mathbf{r}\mathbf{e}^T - (p/q)\mathbf{r}A^T\mathbf{s}^T\Big) \approx \mu,$$

where the first approximation is about rounding.

[15] It is heavily influenced by `https://eprint.iacr.org/2018/331`. See also `https://estimate-all-the-lwe-ntru-schemes.github.io`.

Using LWR, we move on from $c \in \mathbb{Z}_q^{n+1}$ to $c \in \mathbb{Z}_p^{n+1}$ and, therefore, compress ciphertexts. Another possibility of LWR is that we can interpret rounding errors as secret information and achieve in this way the deterministic KEM functionality. This approach is exploited in NPQCC/NTRUPrime (see 5.2.6).

Rounding can be applied not only to vectors of $\mathbb{Z}_q^n$ but also to elements of a ring $\mathbb{Z}_q[x]/(\phi(x))$ or module $(\mathbb{Z}_q[x]/(\phi(x)))^k$. We can formulate the corresponding RLWR (Ring-LWR) and MLWR (Module-LWR) problems that are rather easily converted into PKE/KEM schemes. But first, we should discuss the basic RLWE/MLWE schemes.

RLWE and MLWE schemes. The standard way of making keys and ciphertexts shorter is to convert matrices and vectors over $\mathbb{Z}_q$ into polynomials from $\mathbb{Z}_q[x]/(\phi(x))$. During this conversion, we must preserve security guarantees. These guarantees are dependent on the choice of a ring $\mathbb{Z}_q[x]/(\phi(x))$ and a distribution χ. One (and actually only one) approved choice is when

- $\phi(x)$ is an mth cyclotomic polynomial which degree is $n = \varphi(m)$;
- q is a prime polynomially bounded depending on n;
- $q \equiv 1 \pmod m$;
- χ is a centered Gaussian distibution with a small variance.

It is proved in [144] that the RLWE problem is hard if these conditions hold. More precisely, there exists a quantum reduction from Approximate-SVP on ideal lattices in $\mathbb{Z}[x]/(\phi(x))$ to Decision-RLWE.

Two main instantiations of the approved settings are:

1) $n = 2^k, m = 2n, \phi(x) = x^n + 1$;
2) m is prime, $n = m - 1$, $\phi(x) = (x^m - 1)/(x - 1)$.

The first option is NTT-friendly, the second one is closer to NTRU Classic paramsets.

Let us note that the R.Lizard, KINDI, Saber and uRound2 submissions to NPQCC use unapproved settings in which q is a power of two.

RLWE ecnryption actually repeats the LWE one. Alice generates $s \xleftarrow{\chi_s} R_q$, $a \xleftarrow{R} R_q$, $e \xleftarrow{\chi} R_q$ and calculates $b \leftarrow as + e$. The polynomial s is her private key, the pair (a, b) is public.

To recover a private key from the public one, an adversary must solve the Search-RLWE problem $(a, b \approx as) \mapsto s$ which seems to be hard if parameters are properly chosen.

For comparison,in NTRU an adversary has to solve the problem $h = g/f \mapsto \{f, g\}$ in the same ring R_q. Observe that the NTRU public key is defined without errors and it is twice shorter than the RLWE one.

To encrypt ternary polynomial $m \in R_3$, Bob generates a small random r and calculates

$$u \leftarrow ra, \quad v \leftarrow rb + f(m),$$

the parts of the ciphertext $c = (u, v)$. Alice decrypts c as follows:

$$\hat{m} = f^{-1}(v - us) = f^{-1}(f(m) + re).$$

Decryption works if the polynomial re is enough small.

The MLWE schemes which are based on the MLWE problem are built in a similar way. We only change polynomials from R_q to vectors from R_q^k and the polynomial multiplication $R_q \times R_q \to R_q \colon (f, g) \mapsto fg$ to the inner product $R_q^k \times R_q^k \to R_q \colon (\mathbf{f}, \mathbf{g}) \mapsto \mathbf{f}\mathbf{g}^T$. Since the inner product compresses input data, we can achieve that second parts of public keys and ciphertexts would be shorter than in RLWE.

Reconciliation. The natural way to build the famous Diffie — Hellman protocol in the RLWE settings is the following. Alice and Bob agree on a common random $a \xleftarrow{R} R_q$. Then Alice generates a private key $s_1 \xleftarrow{\chi} R_q$ and sends to Bob the corresponding public key $b_1 \overset{\chi}{\approx} s_1 a$. Bob processes in the same way. His objects are marked with index 2. Receiving b_2, Alice finds $b_2 s_1 \approx s_1 a s_2$. Receiving b_1, Bob finds $b_1 s_2 \approx s_2 a s_1$. Entities have approximately equal polynomials, but only *approximately*. To complete the protocol, Alice and Bob have to remove a noise from their objects and achieve full coincidence. In [76], it was proposed a novel technique how to do this task. This technique was called *reconciliation*. Since then, reconciliation became very popular, a lot of NPQCC candidates (Ding, NewHope, HILA5, etc.) use it.

Let us start to explain reconciliation from the case when q is even. Define the sets $I_0 = \{0, 1, \ldots, \lfloor q/4 \rceil - 1\}$ and $I_1 = \{-\lfloor q/4 \rfloor, \ldots, -1\}$. For $v \in \mathbb{Z}_q$ and $d = 0, 1$ it holds that $v \in I_d \cup (q/2 + I_d)$ if and only if $\lfloor v \rceil_2 = \lfloor (2/q)v \rceil \bmod 2 = d$. In the same way, $v \in I_d \cup (q/2 + I_{1-d})$ if and only if $\langle v \rangle_2 = d$. Here $\langle \cdot \rangle_2$ is the so-called *cross-rounding* function:

$$\langle v \rangle_2 = \lfloor (4/q)v \rfloor \bmod 2.$$

Observe that for a uniform random v, $\langle v \rangle_2$ does not contain information about $\lfloor v \rceil_2$. (Here it is important that q is even.) Therefore, if $\lfloor v \rceil_2$ is secret, it is not a problem to publish $\langle v \rangle_2$. The main idea of reconciliation is that Alice or Bob can recover secret $\lfloor v \rceil_2$ given additionally to $\langle v \rangle_2$ a number $u \in \mathbb{Z}_q$ which is close to v. A non-informative $\langle v \rangle_2$ becomes informative in the presence of u.

Indeed, define the set $E = \mathbb{Z} \cap [-q/8, q/8)$ and introduce the *reconciliation* function $\mathtt{rec} \colon \mathbb{Z}_q \times \mathbb{Z}_2 \to \mathbb{Z}_2$:

$$\mathtt{rec}(u, d) = \begin{cases} 0 & \text{if } u \in (I_d + E) \bmod s\, q, \\ 1 & \text{otherwise.} \end{cases}$$

It is easy to check that $\mathtt{rec}(u, \langle v \rangle_2) = \lfloor v \rceil_2$ if $u = v + e$ and $e \in E$.

Let $\mathtt{dbl}$ be the algorithm which takes $v \in \mathbb{Z}_q$ and returns $2v + e$. Here e is a random trit such that $\mathsf{P}\{e = 0\} = 1/2$, $\mathsf{P}\{e = \pm 1\} = 1/4$. Using this algorithm, we can adapt reconciliation to work with odd modulus q. In fact, we transfer reconciliation from $\mathbb{Z}_q$ to $\mathbb{Z}_{2q}$.

More precisely, if u and v are close, then so are $2u$ and $\mathtt{dbl}(v)$. To (cross-)round from $\mathbb{Z}_q$ to $\mathbb{Z}_2$, we first lift v to $\mathbb{Z}_{2q}$ using $\mathtt{dbl}$ and then apply the appropriate rounding functions. For a uniform random v, rounded bits of different types again do not contain information about each other. To reconcile, we place in $\mathtt{rec}$ the doubled u.

Let us illustrate how reconcilation can be used to provide the KEM functionality. In our example, v is a polynomial from R_q, not a scalar. The functions $\mathtt{dbl}$, $\mathtt{rec}$ are easily extended from elements of $\mathbb{Z}_q$ to such polynomials by applying in the coefficient-wise manner. We suppose that q is odd. We continue to use the previously introduced Diffie — Hellman settings.

Receiving $b_1 \approx a s_1$, Bob additionally to b_2 generates $v \overset{\chi}{\approx} b_1 s_2$. Then he calculates

$$v' \leftarrow \mathtt{dbl}(v), \quad m \leftarrow \lfloor v' \rceil_2, \quad d \leftarrow \langle v' \rangle_2.$$

Bob sends to Alice the ciphertext (b_2, d) and saves m as a session key. Alice recovers this key:

$$\hat{m} \leftarrow \mathtt{rec}(2b_2s_1, d).$$

Reconcilation works since b_2s_1 is close to v.

Let us emphasize that the second part of the ciphertext is the binary polynomial d although in the classic RLWE cryptography this part is q-valued. So, reconcilation shortens ciphertexts almost twice.

Other schemes. Another LWE-related problem is IRLWE (Integer RLWE) introduced in [60]. In this problem polynomials $f(x) \in \mathbb{Z}[x]/(\phi(x))$ are replaced with their values $f(q)$ mod p. Here q is prime and $p = \phi(q)$.

The IRLWE problem was used in NPQCC/ThreeBears. The problem and corresponding schemes seem to be experimental, they do not pretend to be immediately applied right now. M. Hamburg, the submitter of ThreeBears, writes: "*One of our goals ... is to encourage exploration of potentially desirable but less conventional designs.*"

NPQCC/Titanium uses another problem: PLWE (Polynomial-LWE). In PLWE, sets of polynomials with bounded degrees and a special polynomial multiplication are used. The multiplication restricts degrees of products. The new problem also seems to be a bit experimental.

5.2.8 Samplers

Many LBC schemes require sampling from discrete Gaussian distributions. This requirement is usually dictated by a corresponding proof of security (especially it concerns constructions where the worst-case to average-case reduction from well-known lattice problem is provided) so that probability distributions matter.

It is clear that no finite machine is able to sample exactly from a Gaussian distribution, so for practical implementation, one should sample from distribution close to Gaussian. In practice, a statistical distance of less than 2^{-100} is considered sufficient (see [52]).

Even this approximate sampling is computationally heavy, especially for constrained IoT devices. Typical samplers require either floating point operations of big precision or (and) a lot of precomputed values. Sampling often becomes one of the heaviest part (in a sense of computations) in the corresponding cryptosystem, so choosing an effective sampling algorithm (for parameters specific for the cryptosystem) is important for overall performance. In some schemes (e.g., the pNE variant of NTRU encryption), Gaussian sampling actually doesn't prevent any known attacks. In such cases using expensive Gaussian sampling becomes a price for ProvSec.

In this subsection, we describe the most effective approaches for the Gaussian sampling, as well as important use-cases of Gaussian sampling in LBC.

Worth to note that LBC schemes actually suppose rather complicated distributions, like Gaussian distributions over finite fields, rings, or lattices. However, samplers for these distributions could be constructed on top of the regular discrete Gaussian distribution over $\mathbb{Z}$.

Definition 5.1 (Discrete Gaussian distribution over integers)**.** *Discrete Gaussian distribution with center $c \in \mathbb{R}$ and Gaussian "width" parameter $s \in \mathbb{R}, s > 0$ over integers denoted as $D_{s,c}$ is defined by the following probability function:*

$$P_{D_{s,c}}(x) = \frac{\rho_{s,c}(x)}{\sum_{y \in \mathbb{Z}} \rho_{s,c}(y)}, \quad x \in \mathbb{Z},$$

where

$$\rho_{s,c}(x) = \exp(-\pi(x-c)^2/s^2).$$

Standard deviation of $D_{s,c}$ is $\sigma = s/\sqrt{2\pi}$. Distribution $D_{s,c}$ is infinite (to both directions) and has so-called low-weight tails. Therefore, practical sampler should ignore tails from some point. For a desired statistical distance Δ (e.g. 2^{100}), we can determine a safe *tailcut* so that probability of tails is less than Δ, using the following inequality:

$$P_{D_{s,c}}(x \notin [c - t\sigma; c + t\sigma]) \leqslant \frac{1}{t}\exp(-t^2/2).$$

Many sampler algorithms assume calculation of probabilities or cumulative distribution function (CDF), in particular calculating of $\rho_{s,c}$. To keep statistical distance Δ small, calculations should be of precision less than Δ. High-precision arithmetic is usually implemented as a software library, which is much slower than hardware floating point implementations.

Rejection sampling. Rejection sampling is a general method of sampling from any distribution which you can calculate probabilities for. Essentially, for $D_{s,c}$, having Δ and a tailcut t as described above, the algorithm works as follows:

1. Sample x uniformly at random from interval $[c - ts, c + ts] \cap \mathbb{Z}$.
2. Calculate corresponding probability $p = P_{D_{s,c}}(x)$ with precision Δ.
3. Sample r uniformly at random from $[0, 1)$ with precision Δ.
4. Output x in case of $r < p$; reject it and retry the algorithm otherwise.

In [80], an efficient version of this algorithm proposed. The idea of the improvement is that in most cases several most significant bits are enough to decide about the rejection and constructed a kind of lazy variant of the algorithm. It allows using of IEEE standard double precision arithmetics for practical parameters of LBC cryptosystems (e.g., NTRUSign).

This variant of rejection sampling has low memory requirements (it doesn't use any pre-computed tables), although it comes at a price of expensive computations (high-precision evaluations of Gaussian probabilities and multiple retries).

Inversion sampling. Let's denote CDF of Gaussian distribution as $F(x)$. The algorithm is as follows:

1. Sample r uniformly at random from continuous interval $[0, 1)$ with precision Δ.
2. Find x so that $F(x-1) < r \leqslant F(x)$.
3. Output x.

On step 2, inequality could be solved using analytical methods or precomputed values of CDF. In [163], this method with binary search over precomputed values of CDF is proposed for LBC digital signature schemes. In this case, it doesn't involve floating point calculations, but requires large pre-computed tables.

The Ziggurat method. Essentially, it's an upgrade of the rejection method. It could be seen also as a compromise between expensive calculations of rejection sampling, and using of precomputed data.

The idea is to cover curve $P_{D_{s,c}}(x)$ with some number of rectangles of the same area, and save them as a pre-computed data. Then, sampling is the following:

1. Select rectangle i uniformly at random.

2. Inside a rectangle, select a point $p = (x, y)$ uniformly at random.

3. Output x in case of p lies under the curve $P_{D_{s,c}}(x)$; reject and retry algorithm otherwise.

In [52], an adaptation of the general Ziggurat method for discrete Gaussian over integers is proposed for LBC schemes.

Ziggurat method is considered the most effective for some set of parameters, especially for big width s. It provides a good balance between requered precomputed data and expensive high-precision calculations.

The Knuth-Yao sampler. An interesting alternative considered in [15] is Knuth-Yao algorithm, which is designed exactly for discrete distributions. The idea is that having a binary representation of probabilities $p_x = P_D(x), x \in \mathbb{Z}$, we can construct a binary tree binary tree (when implementing the algorithm one uses a table) called a discrete distribution generating (DDG) tree. At each level, we will choose some vertices to be internal, and some to be leaves. The number of leaves at level i is the number of values for a such that the i-th binary digit of p_x is a one. Each leaf is labeled with a different value for x. To sample from the distribution one walks down the tree from the root using one uniform bit at each step to decide which of the two children to move to. When one hits a leaf one outputs the integer label for that leaf.

An advantage of Knuth-Yao sampler is that it's very fast, and uses as little of random bits as possible. At the same time, it requires to store the large precomputed table of probability values. One might think that the algorithm requires more-or-less the same storage as the inversion method (which stores a table of cumulative probabilities). However, the Knuth-Yao table has many leading zeroes and there is no reason to store them directly. So one can expect to reduce the storage for the table.

Comparison of sampler methods. The effectiveness of each of methods depends on the parameters of the distribution and possibilities of the platform the sampler is to be implemented for.

Good performance comparison (benchmark) is provided in [52]. Some analysis of effectiveness is done in [87].

As a conclusion of this overview, we provide some guidance in choosing of sampling method:

1. Knuth-Yao algorithm is the fastest method, however, requires a lot of memory (e.g. up to 1 Mb for some practical parameters, see [15]).

2. In the case of small parameters and devices with sufficient memory (an extra 1–2 kByte), the Knuth-Yao algorithm is still a good choice.

3. If the required Gaussian parameter s is big (more than 500) and/or the memory is extremely constrained, then the best option is the Ziggurat.

5.2.9 Lattice-based signatures

GGH. One of the early proposals for a signature scheme was GGH ([99]). In this scheme, a message digest is mapped to a point $m \in \mathbb{Z}^n$. Also, there's a lattice $\mathcal{L} \subset \mathbb{R}^n$ with two bases: short, almost orthogonal one, which serves as a secret key, and another "random" basis, which serves as a public key.

A signature of the message m is any lattice point $s \in \mathcal{L}$ which is close enough to m (Euclidian distance between s and m is less than a pre-specified bound).

In order to find such s, a signer uses the secret short basis (e.g. using Babai's rounding algorithm, see [16]). And it's hard to find such s without knowing a good basis (CVP problem, see 5.2.2).

Later, the NTRUSign scheme was proposed, which is an efficient instantiation of GGH idea using NTRU lattices.

While the GGH scheme looks like a clear construction built directly on top of the well-known lattice problem, it was not properly elaborated by their authors and was later broken ([104]). It turned out that each produced signature s leaks information about the secret basis, and a long enough transcript of signatures reveals the secret key. The fact is that the point s lies in interior of so-called *fundamental parallelepiped* associated to private basis (fundamental parallelepiped for basis $B = \{b_i\}$ is a set of points $\{\sum_{i=1}^{n} \epsilon_i b_i \mid |\epsilon_i| < 1/2\}$). For example, in case of using Babai's rounding, signature points are uniformly distributed in fundamental parallelepiped. This leads to a revealing of the shape of the parallelepiped and finding a secret basis.

Preimage-sampler signatures. Significant efforts were taken to fix GGH construction in a way that distribution of signature points s doesn't reveal any info about secret basis. E.g., in [96] it's proposed to use randomized version of Babai's rounding, so that distribution of s will be discrete Gaussian over the lattice, with center m and some deviation (which doesn't depend on the shape of the basis, but only on its "shortness"). For this randomized Babai's rounding algorithm, discrete Gaussian sampler over integers is required. The efficiency of this scheme is weakened due to the necessity of sampling from Gaussian distribution, what's more with varying center and "width". Later, in [163] it's proposed an improved version of this approach, so that sampling algorithm is highly-parallelized and required Gaussian distribution has a fixed "width".

It's interesting that this fixed signature generation algorithm in [96] is wrapped up into novel abstraction called *one-way preimage sampleable trapdoor functions (PSF)*. PSF is a surjective function which is easy to evaluate. At the same time, having a trapdoor, it allows sampling (with some distribution, e.g. Gaussian) preimages for the image. PSF could be seen as an extension of the concept of *one-way trapdoor permutation*. Indeed, permutation is a PSF with a domain consisting of a single member, with sampling probability 1. Moreover, it's shown that PSF is a safe replacement of permutations in classical ProvSec signature schemes (which are called *Full-Domain Hash* schemes, or FDH).

PSF is instantiated on lattices as perturbing a lattice point s with some small error e: $m = f(s, e) = s + e$. For some choice of parameters, it's easy to see that there're several preimages of given m. Then, preimage sampling for m using short basis (trapdoor) consists of applying of randomized Babai's rounding algorithm, which produces (s', e') according to some distribution (specifically, s' has the discrete Gaussian distribution over lattice with center m).

Then, the signature scheme could be described just as FDH scheme built on top of specific PSF.

The Modular Lattice Signature Scheme. In [111] another idea of how to hide secret basis from signature transcript is proposed, based on modular properties of the coordinates of lattice vectors. It could be seen as an upgrade of the GGH scheme, where instead of lattice point closest to a message digest we provide a lattice point which is congruent to message digest

modulo some prime number.

Let's give a high-level rough description of the scheme. There's a public lattice $\mathcal{L} \subset \mathbb{Z}^n$ with short secret basis. A message digest is a point $m \in \mathbb{Z}_p^n$ for some public small prime p. Also, there's some public region $R \in \mathbb{Z}^n$ is fixed.

The signature of m is any point $s \in \mathcal{L} \cap R$ so that $s \equiv m \bmod p$. It could be shown that without knowing a short basis it's hard to produce a point which meets both conditions (congruence modulo p, and lying in R).

At the same time, signer uses short basis $B = (b_1, b_2, \ldots, b_n)$ to produce a signature in the following way:

1. Choose a random lattice point $s_0 \in \mathcal{L} \cap R$.
2. Micro-adjust the point s_0 using reduction of the short basis modulo p ($B_p = B \bmod p$) to meet the following congruence condition:
$$s_0 + vB_p \equiv m \bmod p.$$
3. Find a small vector $v \in \mathbb{Z}_p^n$ is found by solving above equation.
4. Build new lattice point $s = s_0 + vB$. Notice, that s also meets the congruence condition. Moreover, since v and B are short, vector vB is also short, and s will lie in R with a significant probability.
5. If s lies in R, output s. Otherwise, return to Step 1.

Additionally, effective instantiation on NTRU lattices, called NTRUMLS, was provided. The scheme does not require any expensive sampling (like Gaussian).

In [111], some parameter sets for NTRUMLS are provided. For example, for 126-bit security level, the proposed parameters yield that a public key consists of 1579 bytes and a signature consists of 1486 bytes. Parameters of the basic NTRU lattice: $n = 563$, $p = 3$, $\log_2 q = 16$.

Later, development of modular lattice signature appeared in [112]. The main modification is the usage of bimodal Gaussian distribution instead of a uniform one, which allows to further reduce signature size.

NPQCC/pqNTRUSign. The scheme of the NPQCC/pqNTRUSign submission is essentially described above (NTRUMLS). The submission provides two options:

- original NTRUMLS, based on uniform sampling, with $\approx 11264B$ signature size;
- advanced NTRUMLS, based on bimodal Gaussian sampling, with $16384B$.

Public key size for both options is $16384B$. NTRU parameters is the same: $n = 1024, p = 3, q = 2^{16} + 1$.

Combining DS and PKE. For goals of IOTA, it's especially interesting if we could combine DS and PKE schemes under single lattice parameters and single key-pair. It seems that the most promising platforms for this goal are preimage-sampler signatures and NTRUMLS. From what we see now, there are prerequisites for combining the same parameters and even the same keys for PKE and DS. However current schemes do not allow to do this directly. For example, polynomial f, which defines (along with polynomial g) NTRU lattice, is chosen differently for NTRUMLS and NTRUEncrypt. But we expect that there's a way to eliminate these differences.

To answer the question, additional research is required.

5.2.10 Conclusion

Here we give answers to the questions asked in Section 5.1.

1. *How short keys and ciphertexts can be?*

   In the main LBC schemes, the lengths of public keys and ciphertexts are dependent on a dimension n and modulus q and are about $n \log q$ bits. We should focus on the ranges $500 \leqslant n \leqslant 1500$ and $2000 \leqslant q \leqslant 15000$ and, therefore, our lengths are of about $700 \div 2600$ bytes. Lengths are doubled in some schemes, for example, in RLWE.

   For comparison, the most compact keys are achieved in the classic ECC (Elliptic Curve Crypto) platform. Their lengths are about $64 \div 128$ bytes (two coordinates of a point on a reasonable safe elliptic curve). LBC loses ECC, but its keys and ciphertexts are quite compact compared to other platforms.

2. *How fast the PKE algorithms can be?*

   LBC algorithms are very effective, this is one of the strengths of the platform. For the best schemes, encryption and decryption require one multiplication of polynomials, which can be done in about $n \log n$ operations modulo q or in about $n \log n (\log q)^2$ bit operations. For comparison, in ECC, the number of bit operations for calculating scalar points (it is the main algorithmic primitive) grows cubically depending on a bit dimension of an underlying curve. For clarity, the latter dimension is much smaller than n (usually from 256 to 512).

3. *How much can we trust the platform given the history of cryptoanalytic attacks?*

   LBC is represented by two large branches: NTRU and LWE. The NTRU branch has been well tested for the past 20 years. No serious attacks on NTRU encryption were found over these years. The classic NTRUEncrypt scheme is stable, it is under international standardization. The LWE branch is relatively new (less than 10 years), but it is actively developing. No serious attacks on LWE encryption were found either.

   The LBC platform is most widely represented in NPQCC. This indicates a great confidence in the platform.

4. *Can we construct DS algorithms using PKE long-term parameters?*

   LBC supports DS schemes although DS algorithms are not so convenient as their PKE/KEM counterparts. Several serious vulnerabilities were discovered until recently in the historically first NTRUSign scheme. The scheme has been upgraded several times and in recent years NTRUSign (as well as its main alternatives) seem to be enough stable.

   In principle, PKE and DS schemes can use the same long-term parameters, although the DS parameters are somewhat heavier.

5. *Can we construct DS algorithms using PKE keypairs?*

   The question requires further research. We have not seen the combining of keypairs in LBC.

6. *If we can construct suitable DS algorithms, how fast they can be and how short their signatures can be?*

   DS algorithms are more cumbersome than their PKE/KEM counterparts. In the first place, this is about signature generation algorithms where for now it is necessary to use iterative algorithms. Nevertheless, DS algorithms are still quite effective. The lengths of DS keys and signatures are comparable to the lengths of PKE keys and ciphertexts.

7. *How appropriate is the ternary logic?*

   Ternary logic fits perfectly with LBC. In fact, ternary logic is even more convenient than binary. This is because LBC mainly uses the centered representation $\mathbb{Z}_q$ ($\mathbb{Z} \cap [-q/2, q/2)$) and in this connection, it is important that plaintexts would be also centered, i.e. represented by numbers in an interval $\mathbb{Z} \cap [-a, a]$. The case $a = 1$ is the most convenient, and it just corresponds to the ternary logic.

5.3 Code-based schemes

5.3.1 Introduction

Code-based cryptography includes schemes constructed on top of error-correcting codes (or simply *codes*). This could sound paradoxically, since codes and ciphers are, in a way, opposite transformations: ciphers try to hide preimage from an image as better as possible, while codes vice-versa allow recovering preimage even if the image is impaired by some noise (errors). However, turns out that this platform is able to generate very strong, quantum-resistant systems, especially PKE primitives.

The main idea for PKE is that in order to encrypt a message, sender encodes it with a specially constructed code with the public random-looking description, and applies random errors. In order to decrypt, recipient applies decoding algorithm known only to him. And, it turns out, that it's impossible (computationally hard) to remove errors without knowing this secret decoding algorithm. This problem (decoding from so-called random linear code) is proved to be NP-complete, so most code-based cryptosystems have solid ProvSec guarantees.

Besides PKE, CBC platform also provides some signature schemes, although they are not so efficient.

5.3.2 Basics of code theory

In this section, some basic notions of linear codes theory required for understanding the following material are provided.

In a broad sense, a code is an injective mapping of messages (called *words*) into another representation — *codewords*. An algorithm of applying this mapping is called *encoding*, an algorithm to invert this mapping (find the word from a codeword) is called *decoding*.

Among the broad class of codes, we're interested in a special subclass, called *linear codes*. The property of linear codes is that any linear combination of codewords is also a codeword. It's easy to describe linear codes in terms of vectors and linear vector transformations — matrices.

The overwhelming majority of codes used in cryptography are linear, so from now on let's use a term *code*, implying a linear code.

Let us have a finite field $\mathbb{F}$ of order q, and vector space $\mathbb{F}^n$ over the field, with some norm $\|\cdot\|$ (typically, it's a Hamming weight). Then, we call $\|a-b\|$ a *distance between vectors* a and b.

Then, we define $[n,k,d]$-code $\mathcal{C}$ as a k-dimensional subvectorspace of $\mathbb{F}^n$, with minimal distance between vectors d. Vectors which lie in $\mathcal{C}$ are called codewords. Dimension k is called a *dimension of the code*. Dimension n is called a *length of the code*. Distance d is called a *distance of the code* $\mathcal{C}$.

Generator matrix G has size $k \times n$. Rows of G form a basis of $\mathcal{C}$. We can denote code $\mathcal{C} = \langle G \rangle$, if G is a generator matrix of $\mathcal{C}$. *Parity check matrix* H has size $(n-k) \times n$. Rows of H form a basis of orthogonal complement of C. For vector $y \in \mathbb{F}^n$, we call $s = yH^T$ a *syndrome* of y. If y is a codeword, syndrome of y equals zero.

For message $x \in \mathbb{F}^k$, encoding consists of multiplying with G:

$$c = xG.$$

Then, error vector $e \in \mathbb{F}^n$ is added to codeword c:

$$y = c + e.$$

Syndrome of y doesn't depend on message x, but only on error-vector e:

$$s = yH^T = cH^T + eH^T = eH^T.$$

For some codes, there's an efficient algorithm to find e of weight less than t by syndrome s, and which allows decoding (finding of x). However, this is an NP-hard problem in general (for random codes).

Goppa codes. Let us provide a sketchy description of Goppa codes, which are of special interest in cryptography.

Goppa code $\Gamma(L, g(z))$ is parametrized by two objects:

- Goppa polynomial $g(z)$, which is a polynomial of degree t over the field $\mathbb{F}_{q^m}$;
- accessory set $L = \{\alpha_1, \ldots, \alpha_n\} \subset \mathbb{F}_{q^m}$, so that $g(\alpha_i) \neq 0$ for all i.

Definition 5.2 (Goppa code)**.** *Goppa code* $\Gamma(L, g(z))$ *is a set of vectors* $c \in \mathbb{F}_q^n$ *so that*

$$\sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} \equiv 0 \bmod g(z).$$

Easy to see, that $\Gamma(L, g(z))$ is actually a subvectorspace, and Goppa codes are linear.

It could be shown that parity-check matrix of $\Gamma(L, g(z))$ could be calculated as $H = CXY$, where

$$C = \begin{pmatrix} -g_t & -g_{t-1} & -g_{t-2} & \ldots & -g_1 \\ 0 & -g_t & -g_{t-1} & \ldots & -g_2 \\ & \ldots\ldots\ldots\ldots & & & \\ 0 & 0 & 0 & \ldots & -g_t \end{pmatrix},$$

$$X = \begin{pmatrix} \alpha_1^{t-1} & \alpha_2^{t-1} & \alpha_3^{t-1} & \dots & \alpha_n^{t-1} \\ \alpha_1^{t-2} & \alpha_2^{t-2} & \alpha_3^{t-2} & \dots & \alpha_n^{t-2} \\ & \dots\dots\dots\dots & & & \\ \alpha_1 & \alpha_2 & \alpha_3 & \dots & \alpha_n \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix},$$

$$Y = \begin{pmatrix} h_1 & 0 & 0 & \dots & 0 \\ 0 & h_2 & 0 & \dots & 0 \\ & \dots\dots\dots & & & \\ 0 & 0 & 0 & \dots & h_n \end{pmatrix}.$$

Then, the corresponding generator matrix G could be calculated from H.

For efficient decoding algorithm, and explanations how Goppa codes work, we recommend looking into [49].

5.3.3 Hard problems

Code with (uniformly) random generator matrix of specific size is called *random code.*

Problem (General Decoding problem, GD)**.** Given a random $\mathcal{C}$ and a vector $y \in \mathbb{F}^n$, find the nearest codeword, i.e. $c \in \mathcal{C}$ for which $\|y - c\|$ is minimal.

If it's guaranteed that there's a nearest codeword on a distance bounded by some maximum value β, we talk about *Bounded Distance Decoding problem (BDD).*

Easy to see, that BDD is equivalent to the following problem.

Problem (Syndrome Decoding problem, SD)**.** Given a $(n-k) \times n$ matrix H, a vector $s \in \mathbb{F}^{n-k}$ and a nonnegative integer w, find a vector $e \in \mathbb{F}^n$ of Hamming weight $\leqslant w$ such that $eH^T = s$.

It's shown in [25] that decisional version of SD is NP-complete in case of binary vectors and matrices (over the field $\mathbb{F}_2$). In [21], the result is extended to fields of arbitrary order q.

Intuitively, we can explain the hardness of the problem in the following way:

1. Equation $eH^T = s$ defines a system of $n-k$ (independent) linear equations in n variables (from field $\mathbb{F}_q$).
2. There are q^k solutions to the system, and only some (a little part) of them has weight w.
3. To find a solution with weight w, we have to look over them one by one, which takes $O(q^k)$ operations.

This means that SD, BDD (and moreover, GD) are NP-hard in case of Hamming-weight norm.

Worth to note, that this means impossibility to solve this problem in the worst case, not in the average case. Moreover, we use not random, but specific codes in cryptosystems (like permuted Goppa-code in McEliece PKC), for which this problem is not proven to be NP-hard.

However, it is widely believed that the following problem is hard.

Problem (Goppa Code Distinguishing problem, GCD)**.** Given a $(n-k) \times n$ binary matrix H, find whether H is a parity check matrix of a $[n, k]$-Goppa code or of a random $[n, k]$-code.

The main motivation for introducing the GCD problem is to formally relate GD problem to the security of the McEliece public-key cryptosystem (e.g. this is done for DS scheme in [57]). The hardness of the GPD problem is not proven, moreover, there's some progress in solving it ([88]).

5.3.4 McEliece PKE scheme

McEliece PKE scheme.McEliece PKE is based on the idea of using the following one-way function with trapdoor: encoding with a linear code defined by public generator matrix G and applying of random errors. It's indeed a one-way function since G looks like a random matrix, and GD problem is NP-complete. At the same time, the secret key owner has a trapdoor — knowledge of a specific structure of the code $\langle G \rangle$ (in original McEliece construction — an instance of permuted Goppa code), which gives an effective decoding (error-correcting) algorithm.

Secret key. A secret key consists of:

- $D_{\mathcal{C}}$ — decoding algorithm for random $[n, k, d]$-code $\mathcal{C}$ with error-correction ability t;
- S — random invertible $k \times k$-matrix over $\mathbb{F}_p$, called *row-scrambler*;
- P — random $n \times n$ permutation matrix over $\mathbb{F}_p$.

In case of Goppa codes, which is the original McEliece proposal, for $D_{\mathcal{C}}$ it's sufficient to store Goppa polynomial $g(z)$ and accessory set L. $g(z)$ is a polynomial of degree t over field $\mathbb{F}_{p^m}$. L has $n \leqslant p^m$ elements from $\mathbb{F}_{p^m}$. Goppa polynomial and accessory set completely define code $\mathcal{C}$ and decoding algorithm $G_{\mathcal{C}}$.

Public key. Public key is a pair (G^{pub}, t). G^{pub} is evaluated from secret generator matrix G of the code $\mathcal{C}$ by multiplication it with secret row scrambler and permutation matrices:

$$G^{pub} = SGP.$$

G^{pub} is a $k \times n$-matrix over $\mathbb{F}_p$.

Public key also includes error-correction ability t of the code.

Plaintext. Plaintext are elements of $\mathbb{F}_p^k$.

Ciphertext. Ciphertexts are (some specific) elements of $\mathbb{F}_p^n$. Worth to note, that share of ciphertexts among all vectors of $\mathbb{F}_p^n$ is low, but it's hard (without knowing the secret key) to distinguish valid ciphertext from random vectors.

Encryption. To encrypt a message x, sender executes the following procedure:

1. Random vector $e \in \mathbb{F}_p^n$ of weight t is generated.
2. Ciphertext is computed as encoding message x with code $\langle G^{pub} \rangle$ and applying error vector e:
$$y = xG^{pub} + e.$$

It's an extremely fast operation (just a matrix multiplication).

Decryption. To decrypt a ciphertext y, one needs to do the following actions:

1. Reverting permutation:
$$c = yP^{-1}.$$

2. Applying decoding algorithm:
$$m' = D_{\mathfrak{C}}(c).$$

3. Reverting row scrambler:
$$m = m'S^{-1}.$$

Difficulty: matrix multiplication and decoding procedure (which is pretty fast).

Security. It's an OW-CPA secure scheme (on assumption of GD and GCD are hard). By applying specific conversions one can obtain IND-CPA2 scheme from it.

Attacks. There're two directions of attacks:

- Structural Attack: Recover the secret transformation and the description of the secret code from public key (G^{pub}, t). Depends on the class of codes and the secret transformation used;
- Ciphertext-Only, or Direct Attack: Recover the original message from the ciphertext and the public key. Corresponds to solving the GD problem.

Both attacks have no successful implementations in case of Goppa codes usage.

In this overview only direct attacks on $[n, k, d] - code$ over $\mathbb{F}_p$ with error-correction ability $t = \lfloor d/2 \rfloor$ are described.

Guessing error vector (Naive Syndrome Decoding). Having a syndrome $s \in \mathbb{F}_p^{(n-k)}$, we can try to guess error vector e as a vector of size n with weight $\leqslant t$ and compare eH^T with s. The number of error vectors to try (difficulty of attack):

$$\sum_{i=1}^{t}(p-1)^i\binom{n}{i} > (p-1)^t\binom{n}{t}$$

Information set decoding (ISD). The most effective tactics. Idea is to guess k correct and independent positions out of n (where no errors occurred). The probability that the k positions you choose are correct is about $(1 - \frac{t}{n})^k$, and the Gaussian elimination you apply involves k^3 steps, so the expected workload is:

$$\frac{k^3}{(1-\frac{t}{n})^k}$$

One of the improved versions of information set decoding is *Bit Swapping*. For more information, look into [124].

Asymptotic complexity of ISD is provided in [56].

Weak point — multiple encryptions of the same message. One of the disadvantages of the system is that it is not safe for the use of sending the same message several times with the same encryption. However, this situation is not supposed in IOTA.

5.3.5 The Niederreiter PKE scheme

It's actually a dual variant of McEliece scheme, having the same security, but a bit different effectiveness. The idea is to encode plaintext not as a codeword, but as an error-vector. Then, ciphertext becomes a syndrome instead of a codeword. The public key is parity-check matrix H. To map message $x \in \mathbb{F}^k$ to error-vector $e \in \mathbb{F}^n$ of weight t, special invertible algorithm ϕ needed: $e = \phi(x)$.

In comparison to the McEliece scheme, it has the following advantages:

- compress public key: it allows to publish public key (parity check matrix) H^{pub} in a systematic form, so it has a size $(n-k) \times k$ (compare to $k \times n$ generator matrix in case of McEliece);
- shorted cipher text: now it's a syndrome, which has a size $(n-k)$ (instead of n-length codeword).

Disadvantages are:

- slower encryption/decryption: applying ϕ and ϕ^{-1}; however, in case of KEM, this disadvantage disappears, since we can encrypt a random t-weight vector e, and then use hash function to derive transmitted key from e.

5.3.6 The CFS signature scheme

CFS is based on Niederreiter PKE. The idea is to treat message digest as a syndrome, and signature of the message is the corresponding error vector. The problem with this approach is that not all syndromes (but only a small part of them) could be decoded. Easy to see, that for $[n, k, d]$-code with error-correction t, the probability of random syndrome to be decodable is about $\frac{1}{t!}$.

If t is not big (e.g. 9 or 10), the following signature scheme is possible:

1. Choose a random *nonce* of sufficient length.
2. Compute the message digest of a message x with the nonce: $s = hash(x \parallel nonce)$.
3. Try to decode syndrom $s \in \mathbb{F}^n$, i.e. find $e \in \mathbb{F}^k : eH^T = s$.
4. If e of weight t is found, return pair $(e, nonce)$ as a signature. Otherwise, go to step 1.

Signature generation requires look over $t!$ nonces. To make CFS scheme possible, we need a code, which simultaneously:

- has proper security properties (difficulty of GD problem), which depends on, or more exactly, grows with n and t;
- has low t (e.g. $\leqslant 10$), to allow acceptable signature generation time.

It requires to choose parameters of codes in a non trivial way, different from what we choose for PKE schemes. In other words, for the codes correcting such a little number of errors we need to have very long codewords in order to achieve good security. It leads to a huge public

key size. However, signature size is extremely small — it's just a vector e of small weight t, which allows for space-efficient encoding.

E.g., parameters proposed in [57]: binary Goppa code with $n = 2^{16}, k = 2^{16} - 144, t = 9$, which gives 90 bits of security. Public key is parity check matrix H in systematic form, which takes $144 * (2^{16} - 144) \approx 2^{23}$ bits. The signature could be encoded in just 144 bits. This is not so positive feature, since having such a small signature space allows using of brute-force attack, independent on the strength of one-way of function. I.e., for 144-bit signatures we have just complexity of 2^{144} for a pre-quantum attack on finding a signature for a given message digest, or 2^{72} attack on finding a signature for some message digest (birthday attack on syndromes space of size 2^{144}).

Trying to build PKE and DS schemes on the same public key would lead to this disadvantage: for CFS scheme to be secure, we need to use large code size, which in turn leads to a big *ciphertext expansion* (how much ciphertext is bigger than plaintext), and PKE scheme becomes not very performance-effective.

In general, the CFS scheme has the following weaknesses:

- a lack of existential unforgeability under adaptive chosen message attacks (EUF-CMA);
- bad scaling of parameters.

More precisely:

- the parity check matrix of high rate Goppa code can be distinguished from a random matrix and thus the CFS signature scheme is insecure under the EUF-CMA ([A distinguisher for high-rate McEliece cryptosystems]());
- the decoding attack complexity A is only a polynomial function of the key size with small power, that is $A \approx keysize^{t/2}$; because we should keep t small (up to 12), we need to significantly increase key size;
- with small t, ciphertext expansion is big.

There's a single stable CBC signature submission to NPQCC — pqsigRM.

The idea of the pqsigRM is to implement CFS scheme with another class of codes — Reed-Muller. These codes allow complete decoding. However, the decoding method does not guarantee the exact error correction, which is not a problem for a signature scheme, but will be a problem for a PKE scheme. Also, simple replacement of Goppa code with RM code leads to several structural attacks. For more information see [136]. In other words, RM codes is a not a stable solution for cryptosystem, comparing to Goppa codes, and better security analysis is required.

5.3.7 Ternary numeral system for CBC

An overwhelming majority of works in this area corresponds to binary CBC, which means using binary codes, and constructions over fields of cardinality 2. At the same time, applying ternary numeral system, making CBC adapted for IOTA infrastructure, is possible by using fields of cardinality 3. E.g., binary Goppa codes could be substituted with ternary Goppa codes. We just need to make sure that:

– all proofs stay valid in a ternary case;

– ternary constructions are efficient (even if not optimal), at least not less efficient than just emulating of optimal binary constructions.

Binary Goppa codes are optimal, while codes of other cardinalities have a twice worse error-correction ability. If $g(z)$ — Goppa polynomial of degree t, then error correction of binary separable Goppa code is t, while it's only $t/2$ in ternary (and any other) case. This is obviously critical for error-correction coding, but it's not clear how it affects security and effectiveness of the cryptosystem. Intuitively, low error weight leads to easier guessing-error-vector attack, which should be compensated with an increase of n, and consequently, an increase of public key size, slower encryption/decryption. Ciphertext expansion will be affected too: for Goppa codes, in case of McEliece variant, it equals to $R = \frac{n}{k} = \frac{n}{n-mt}$.

5.3.8 Submissions to NPQCC

There're three effective (in a sense of short ciphertexts and public keys) implementations of McEliece/Niederrieter PKE on Goppa codes:

– Classic McEliece ([27]);

– NTS-KEM ([9]);

– BIG QUAKE ([65]).

These schemes have similar performance characteristics: small ciphertext expansion, but large public keys.

There's also very effective (small ciphertext expansion, and small public key) KEM scheme which is based on the intersection of CBC and LBC — LAKE ([12]). Signatures are represented by a single candidate — CFS-like signature scheme pqsigRM ([136]).

DS submission (pqsigRM) is already reviewed in 5.3.6.

Then, below let's review only PKE/KEM submissions. We consider mainly two performance characteristics: public key size, and ciphertext expansion (ratio of sizes of ciphertext and plaintext). The reason is that space efficiency is the most important thing for MAM2 (and IOTA in general). Moreover, since speed (of key generation, encryption, decryption) of all CBC constructions is quite high, we will not pay much attention to it.

Classic McEliece. It's a Niederrieter variant, with Goppa codes used. Two sets of parameters are defined:

1. `mceliece6960119`: $n = 6960$, $t = 119$, $k = n - mt = 5413$, $q = 2^{13}$, field $\mathbb{F}_q$ polynomial is $f(z) = z^{13} + z^4 + z^3 + z + 1$.

2. `mceliece8192128`: $n = 8192$, $t = 128$, $k = n - mt = 6528$, $q = 2^{13}$, field $\mathbb{F}_q$ polynomial is $f(z) = z^{13} + z^4 + z^3 + z + 1$.

Both parameter sets provide 2^{512} security level.

The goal of Classic McEliece submission is KEM, so it uses the following KEM construction over Niederrieter PKE:

– uniform random input e, not using of ϕ mapping;

– compute the session key as a hash of e;

– ciphertext includes (besides original Niederrieter PKE ciphertext) a "confirmation": another cryptographic hash value of e.

The public key size is $k(n-k)$ bits (parity check matrix in systematic form), which means:

1. 1047319 bytes (equals to 1746000 trytes) for `mceliece6960119`.

2. 1357824 bytes (equals to 2263000 trytes) for `mceliece8192128`.

KEM ciphertext expansion could be calculated as $(SizeOfSyndrome + SizeOfConfirmation)/SizeOfHashValueOfErrorVector$, which equals to:

1. $\frac{226(byte)}{32(byte)} \approx 7$ for `mceliece6960119`.

2. $\frac{240(byte)}{32(byte)} \approx 8$ for `mceliece8192128`.

BIG QUAKE and NTS-KEM have similar to McEliece characteristics.

But they provide parameter sets for 2^{128} security level, with smaller keys.

BIG QUAKE. The key advantage of this proposal is reducing the key size by a moderate factor l in the range $[3 \dots 19]$. This is obtained by using binary quasi-cyclic Goppa codes of order l instead of plain binary Goppa codes.

There's a loss in structural attack complexity, but since generic attacks on Goppa codes are much more efficient than structural, this loss is affordable.

Public key size is 25482 bytes (equals to 42470 trytes)

Ciphertext expansion of KEM is $\frac{201(bytes)}{32(bytes)} \approx 6.3$.

NTS-KEM . Public key 319488 bytes (equals to 532480 trytes).

Ciphertext expansion of KEM is $\frac{128(bytes)}{32(bytes)} \approx 4$.

LAKE. LAKE is located on the intersection of CBC and LBC. The whole scheme is McEliece, however usage of LRPC codes with Rank metric involve lattice constructions.

The idea appeared at first time in [105]. Authors highlighted that the cryptosystem can be seen as an equivalent to the NTRU cryptosystem, in a rank metric context.

The good feature of LRPC codes that they allow to choose non-binary fields (ternary in our case) without loss in effectiveness/optimality.

Public key size is:

– 394 (bytes) (equals to 656 trytes) for 2^{128} security level;

– 590 (bytes) (equals to 983 trytes) for 2^{192} security level;

– 789 (bytes) (equals to 1315 trytes) for 2^{256} security level.

Ciphertext expansion of KEM is:

– $\frac{394(bytes)}{64(bytes)} \approx 6.1$ for 2^{128} security level;

– $\frac{590(bytes)}{64(bytes)} \approx 9.2$ for 2^{192} security level;

– $\frac{789(bytes)}{64(bytes)} \approx 12.3$ for 2^{256} security level.

Speed is more-or-less the same as of the most efficient LBC submissions, based on NTRUEncrypt.

5.3.9 Conclusion

Advantages of employing CBC platform are:

- very fast encryption and decryption;
- solid ProvSec guarantees;
- stability, long story of unsuccessful cryptoanalysis in case of conservative parameters (McEliece scheme on Goppa codes appeared about 40 years ago, and stays unbroken in almost the original version).

Then main disadvantage is:

- extremely large public keys;

Additional disadvantages come from our potential goal of combining DS and PKE:

- at least on present-day, it's not possible to use the same parameters for DS and PKE scheme;
- DS constructions are rather inefficient.

5.4 Multivariate-based schemes

5.4.1 Introduction

Multivariate-based scheme (MB-scheme) is a cryptographic scheme whose public key is a set of (usually) quadratic polynomials over a finite field $\mathbb{F}_q$. These schemes can serve to implement various cryptographic primitives such as asymmetric encryption, signature, key encapsulation mechanism (KEM), zero-knowledge authentication scheme.

The public key of MB-scheme is in general given by a set of quadratic polynomials

$$\begin{aligned} p_1(x_1,\ldots,x_n) &= \sum_{1\leqslant i\leqslant j\leqslant n} a_{ij}^{(1)} x_i x_j + \sum_{1\leqslant i\leqslant n} b_i^{(1)} x_i + c^{(1)}, \\ &\ldots \\ p_m(x_1,\ldots,x_n) &= \sum_{1\leqslant i\leqslant j\leqslant n} a_{ij}^{(m)} x_i x_j + \sum_{1\leqslant i\leqslant n} b_i^{(m)} x_i + c^{(m)} \end{aligned}$$

with all coefficients and variables in $\mathbb{F}_q$ [77, 108]. A public key is supposed to be hardly distinguishable from a random set of quadratic polynomials and therefore be difficult to invert.

The public key defines the quadratic map $\mathcal{P} = (p_1,\ldots,p_m)$. To use MB-scheme as a cryptographic primitive, one needs a trapdoor built into the public map $\mathcal{P}$. The main idea for the construction of MB-schemes is to represent $\mathcal{P}$ as a composition of maps that can be easily reversed individually, whose composition is hard to reverse without knowing the elementary components. This composition is the private key and defines the trapdoor of MB-scheme. The trapdoor helps to invert the map $\mathcal{P}$, which is a public key. The main challenge in MB-schemes is the selection of the trapdoor.

Most MB-schemes have standard construction $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{S}$, where $\mathcal{S} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ and $\mathcal{T} : \mathbb{F}_q^m \to \mathbb{F}_q^m$ are invertible affine maps, $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ and $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ are quadratic maps.

The idea for the using of standard construction is to choose a map $\mathcal{F}$ (called a central map) which can be easily inverted. After that one chooses invertible affine maps $\mathcal{S}, \mathcal{T}$ to hide the structure of $\mathcal{F}$ in $\mathcal{P}$. A tuple of three maps $(\mathcal{S}, \mathcal{F}, \mathcal{T})$ is the private key.

Generally, the size of the public key is $m \cdot \tau(n)$ field elements [182], where

$$\tau(n) = \begin{cases} 1 + n + \frac{n(n-1)}{2} = 1 + \frac{n(n+1)}{2} & \text{if } q = 2, \\ 1 + n + \frac{n(n+1)}{2} = 1 + \frac{n(n+3)}{2} & \text{otherwise.} \end{cases}$$

The size of the private key is specific for various MB-schemes.

Encryption is the same for all standard constructions. To encrypt a message $\mathbf{x} \in \mathbb{F}_q^n$, one simply computes $\mathbf{y} = \mathcal{P}(\mathbf{x})$ from the public key. Decryption essentially is characterized by the central map $\mathcal{F}$, which is specific for various MB-schemes. To decrypt a given ciphertext $\mathbf{y} \in \mathbb{F}_q^m$, one computes recursively $\mathbf{y}' = \mathcal{T}^{-1}(\mathbf{y})$, $\mathbf{x}' = \mathcal{F}^{-1}(\mathbf{y}')$ and $\mathbf{x} = S^{-1}(\mathbf{x}')$ from the private key.

Signature generation and verification are similar to decryption and encryption respectively. To generate a signature for a message (a hash value) $\mathbf{y} \in \mathbb{F}_q^n$ one computes recursively $\mathbf{y}' = \mathcal{T}^{-1}(\mathbf{y})$, $\mathbf{x}' = \mathcal{F}^{-1}(\mathbf{y}')$ and $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{x}')$ from the private key. The signature of the message $\mathbf{y}$ is given by $\mathbf{x} \in \mathbb{F}_q^n$. To check the authenticity of a signature $\mathbf{x} \in \mathbb{F}_q^n$, one computes $\mathbf{y}^* = \mathcal{P}(\mathbf{x})$ from the public key. If $\mathbf{y}^* = \mathbf{y}$ holds, the signature is accepted, othewise is rejected.

5.4.2 Hard problems

The generic security of MB-schemes is based on the hardness of solving a non-linear system of multivariate polynomial equations, or the following problem:

Problem (Polynomial System Solving, PoSSo)**.** Given m nonlinear polynomials $p_1(x_1, \ldots, x_n), \ldots, p_m(x_1, \ldots, x_n) \in \mathbb{F}_q[x_1, \ldots, x_n]$ with degree of d, find values $x_1^*, \ldots, x_n^* \in \mathbb{F}_q$ such that $p_1(x_1^*, \ldots, x_n^*) = 0, \ldots, p_m(x_1^*, \ldots, x_n^*) = 0$.

We note that the number of terms for one polynomial of the PoSSo problem increases with $O(n^d)$ — and is hence very sensitive to the degree d [182].

The PoSSo problem is proven to be NP-hard and remains so even if it is restricted to a system of multivariate quadratic (MQ) equations [93, 165]. The PoSSo problem with quadratic polynomials is called the MQ-problem for multivariate quadratic.

The hardness of MQ-problem depends on the ratio between the number of variables n and the number of equations m. For a random system of equations, MQ-problem is exponential when $m = O(n)$, but it becomes polynomial when the system is heavily overdetermined (i.e. when $m = O(n^2)$) or heavily underdetermined (i.e. when $n = O(m^2)$). Generally, encryption MB-schemes use an overdetermined system of equations, so $n < m < n^2$, while signature MB-schemes use an underdetermined system, so $m < n < m^2$ [165].

To use MQ-problem as underlying hard problem in MB-schemes, one need a trapdoor built into the public map $\mathcal{P}$. So the security of MB-schemes is not only based on the MQ-Problem, but also on some variant of the Isomorphism of Polynomials (IP) problem. Furthermore, some MB-schemes require the hardness of the PoSSoWN problem [59] and the MinRank problem [173]. The underlying problems are defined as follows:

Problem (Extended Isomorphism of Polynomials, EIP)**.** Give a special class of nonlinear maps $\mathcal{C}$ and a map $\mathcal{P}$ expressible as $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{S}$, where $\mathcal{S}$, $\mathcal{T}$ are affine maps and $\mathcal{F} \in \mathcal{C}$, find a decomposition of $\mathcal{P}$ such that $\mathcal{P} = \mathcal{T}' \circ \mathcal{F}' \circ \mathcal{S}'$ for affine maps $\mathcal{S}'$ and $\mathcal{T}'$, and $\mathcal{F}' \in \mathcal{C}$.

Problem (Polynomial System Solving with Noise, PoSSoWN)**.** Given m nonlinear polynomials $p_1(x_1, \ldots, x_n), \ldots, p_m(x_1, \ldots, x_n) \in \mathbb{F}_q[x_1, \ldots, x_n]$ with degree of d, find values $x_1^*, \ldots, x_n^* \in \mathbb{F}_q$ such that for all f_i, we have $p_i(x_1^*, \ldots, x_n^*) + e_i = 0$, where e_i is error chosen from a certain distribution.

Problem (Minimal Rank, MinRank)**.** Let $m, n, r, k \in \mathbb{N}$ and $r, m < n$. Given $M_1, \ldots, M_k \in \mathbb{F}_q^{m \times n}$, find a non-zero k-tuple $(\lambda_1, \ldots, \lambda_k) \in \mathbb{F}_q^k$ such that the rank of the matrix $\sum_{i=1}^k \lambda_i M_i$ does not exceed r.

There is not much known about the difficulty of the IP/EIP problem. Patarin et al. [158] show that solving IP problem is at least as difficult as the Graph Isomorphism (GI) problem (and perhaps much more difficult) so that IP is unlikely to be solvable in polynomial time. Geiselmann et al. [94] show that EIP problem where $\mathcal{C}$ is the set of the gomogeneous quadratic maps is easy.

The PoSSoWN and MinRank problems have been proved to be NP-hard [8] and NP-complete [172] respectively.

We note that almost no MB-schemes come with worst-case to average-case reductions and very few have full security reductions from the generic NP-hard or NP-complete problem. More usually, there are reductions from the problem of solving the specific type of non-random trapdoor system used in the public key [165].

5.4.3 General constructions

In this subsection, we consider some of the popular MB-schemes. Although these schemes are not used explicitly in PKE and KEM of NPQCC, they may be useful in the contexts of the signature algorithms and the possible progress of PKE/KEM.

HFE. The HFE (Hidden Fields Equations) scheme was proposed by J. Patarin in [155]. This scheme is probably one of the most popular. It can be described as follow.

Let $q = p^e$, where p is a prime and $e \geqslant 1$. Let $\mathbb{E}$ be an extension of the finite field $\mathbb{F}_q$ of degree n and $\phi : \mathbb{E} \to \mathbb{F}_q^n$ be the $\mathbb{F}_q$-linear isomorphism map between the finite field $\mathbb{E}$ and the vector space $\mathbb{F}_q^n$. The central map of HFE is defined a univariate polynomial $F(X) \in \mathbb{E}[X]$ of the following form

$$F(X) = \sum_{0 \leqslant i \leqslant j \leqslant d} \alpha_{ij} X^{q^i + q^j} + \sum_{0 \leqslant i \leqslant d} \beta_i X^{q^i} + \gamma,$$

where $1 \leqslant d \ll n$ is an integer and $\alpha_{ij}, \beta_i, \gamma \in \mathbb{E}$. The special structure of $F(X)$ is chosen such that its multivariate representation over the base field $\mathbb{F}_q$ has only quadratic polynomials.

The public key is given to be $\mathcal{P} = \mathcal{T} \circ \phi \circ F \circ \phi^{-1} \circ \mathcal{S}$. The private key consists of the maps $\mathcal{T}$, $\mathcal{S}$ and a polynomial F.

The encryption process is standard: $\mathbf{y} = \mathcal{P}(\mathbf{x})$. Inversion is accomplished by taking a ciphertext $\mathbf{y}$, computing $\mathbf{y}' = \mathcal{T}^{-1}(\mathbf{y})$, solving $\phi(\mathbf{y}) = F(X)$ for X via the Berlekamp algorithm and then recovering $\mathbf{x} = \mathcal{S}^{-1}(\phi^{-1}(X))$. Its complexity is $O(D^3 + nD^2 \log q)$, where $D \leqslant 2q^d$. Degree d cannot be too high since this would affect the decryption process. Also, d cannot be

too small because this would make HFE vulnerable to message-recovery attacks using Grobner basis techniques [73, 86].

The security of HFE has been thoroughly investigated. The most powerful attacks are exponential in the degree d so it is possible to choose parameters for which HFE is secure. Unfortunately, these parameters also lead to unacceptably slow decryption times [160]. Thus HFE itself has been considered to be impractical.

There are a number of variants of HFE such as HFE- (the minus variant) [155], HFEv (the vinegar variant) [157], HFEv- (the minus and vinegar variant) [161]. The idea of HFE- is to remove r (a small number) of equations from the public map $\mathcal{P}$. The idea of HFEv is to parametrize the polynomial F by adding additional t (vinegar) variables. HFEv- combines ideas of HFE- and HFEv. In the signature context, these variants are more efficient than a plain HFE. In the context of encryption, there are the performance penalties induced by the fact that we will have to re-run the decryption process several times (q^t times for HFEv, and q^r times for HFE-) [160].

HFE and its variants are used as building blocks in the signature algorithms NPQCC/DualModeMS, NPQCC/GeMSS, NPQCC/Gui, NPQCC/HiMQ-3

UOV. The UOV (Unbalanced Oil and Vinegar) scheme was proposed by A. Kipnis at el. in [127, 128]. This scheme is a generalization of the oldest MB-scheme of Patarin [156]. It can be described as follow.

Let $\mathcal{V} = \{1, \ldots, v\}$, $|\mathcal{V}| = v$ and $\mathcal{O}(v+1, \ldots, n)$, $|\mathcal{O}| = o$. We call $x_1, \ldots, x_v$ as the vinegar variables and $x_{v+1}, \ldots, x_n$ as the oil variables. The central map is a set $\mathcal{F} = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))$ of multivariate quadratic polynomials

$$f_k(\mathbf{x}) = \sum_{i \in \mathcal{O},\ j \in \mathcal{V}} \alpha_{ij}^{(k)} x_i x_j + \sum_{i,j \in \mathcal{V},\ j \leqslant j} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in \mathcal{V} \cup \mathcal{V}} \gamma_i^{(k)} x_i + \eta^{(k)},$$

where $\alpha^{(k)}, \beta^{(k)}, \gamma^{(k)}, \eta^{(k)} \in \mathbb{F}_q$ and $m = o$. Note that the vinegar variables are combined quadratically, while the oil variables are only combined with vinegar variables in a quadratic way. To invert the central map $\mathcal{F}$ one choose a vector $\mathbf{v} = (x_1, \ldots, x_v) \in \mathbb{F}_q^v$ of the vinegar variables at random and plug $\mathbf{v}$ into f_k for $k = 1, \ldots, o$. Then, one gets a linear system of o equations with o variables $x_{v+1}, \ldots, x_n$ and solves the linear system by using the Gaussian elimination.

The public key is the map $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$ with an invertible affine map $\mathcal{T} : \mathbb{F}_q^n \to \mathbb{F}_q^n$. The private key consists of the maps $\mathcal{F}$, $\mathcal{T}$ which allow to invert the public key.

UOV can only be used for signature schemes as we need $v \geqslant 2o$ for a secure construction. The signature generation and verification processes are standard.

The main disadvantage of UOV is arguably that its public keys are quite large. For example, with the parameters $q = 16$, $m = 16$, $n = 48$ from [128], we have a public key of 16 kB and the corresponding private key of 1152 B. In addition, UOV has a large ratio between the number of variables and equations (3:1) making the signatures three times longer than the messages (the hash values).

UOV is pretty well studied. The most efficient attacks on UOV have a complexity of $O(q^{v-m-1}m^4) = O(q^{n-2m-1}m^4)$ [46, 127].

Generalizations of UOV are NPQCC/Rainbow and NPQCC/LUOV. The key idea of NPQCC/Rainbow is to use multiple layers of UOV to improve efficiency and reduce the key sizes. NPQCC/LUOV is a simple improvement of UOV that reduces the size of the public keys.

ABC. The ABC (also called Simple Matrix) scheme for encryption was proposed Ch. Tao et al. in [178]. This scheme is constructed from products of square polynomial matrices. It can be described as follow.

Let $s, n, m \in \mathbb{N}$ and $n = s^2$, $m = 2n$. For a given s, n, m one define three $s \times s$ matrices A, B and C of the form

$$A = \begin{pmatrix} x_1 & \dots & x_s \\ x_{s+1} & \dots & x_{2s} \\ \vdots & & \vdots \\ x_{(s-1)s+1} & \dots & x_n \end{pmatrix}, \; B = \begin{pmatrix} b_1 & \dots & b_s \\ b_{s+1} & \dots & b_{2s} \\ \vdots & & \vdots \\ b_{(s-1)s+1} & \dots & b_n \end{pmatrix}, \; C = \begin{pmatrix} c_1 & \dots & c_s \\ c_{s+1} & \dots & c_{2s} \\ \vdots & & \vdots \\ c_{(s-1)s+1} & \dots & c_n \end{pmatrix},$$

where $x_1, \dots, x_n \in \mathbb{F}_q$, $b_1, \dots, b_n$ and $c_1, \dots, c_n$ are randomly chosen as linear combination of elements from the set $\{x_1, \dots, x_n\}$. Define $E_1 = A \cdot B$, $E_2 = A \cdot C$ and let $f_{(i-1)s+j} \in \mathbb{F}_q[\mathbf{x}]$ and $f_{s^2+(i-1)s+j} \in \mathbb{F}_q[\mathbf{x}]$ be respectively the (i, j)-element of E_1 and E_2, where $\mathbf{x} = (x_1, \dots, x_n)$ and $i, j = 1, 2, \dots, s$. Then the central map $\mathcal{F}$ of the scheme is $\mathcal{F} = (f_1(\mathbf{x}), \dots, f_m(\mathbf{x}))$.

The public key is the composed map $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ with two randomly chosen invertible linear maps $\mathcal{S} : \mathbb{F}_q^m \to \mathbb{F}_q^m$ and $\mathcal{T} : \mathbb{F}_q^n \to \mathbb{F}_q^n$. The private key consists of the matrices B, C and the linear maps $\mathcal{S}$, $\mathcal{T}$.

The encryption and decryption processes are standard. Main disadvantages of ABC are the high probability of decryption failures (about q^{-1}) and the high factor between plaintext and ciphertext size. For example, 100 bit security ABC scheme ($q = 2^8$, $s = 10$, $n = 100$, $m = 200$) has the plaintext of 100 B, the ciphertext of 200 B, the public key of 1030 kB, the private key of 70 kB, the probability of decryption failures of 2^{-8}.

For the security, it is known that the structural attack, the min-rank attack, and the linearization attack are available on ABC [108, 152].

To improve security and to reduce the probability of decryption failure, several versions of ABC have been proposed, e.g. the rectangular version [177], the tensor version [162], the cubic version [74]. However, the security for such versions should be studied carefully. For example, it was shown that the security for the cubic version against the linearization attack is almost same to the original ABC [108].

5.4.4 The candidates of NPQCC

Currently, there are the following 9 MB-schemes for round 1 of NPQCC: CFPKM, DME, DualModeMS, GeMSS, Gui, HiMQ-3, LUOV, MQDSS, Rainbow. Of these schemes, there are 8 signature algorithms (DME, DualModeMS, GeMSS, Gui, HiMQ-3, LUOV, MQDSS, Rainbow) and 2 KEM (CFPKM, DME).

For signature algorithms there are 4 HFE-based (DualModeMS, GeMSS, Gui, HiMQ-3) and 2 UOV-based (LUOV, Rainbow), with 2 algorithms (DME, MQDSS) falling outside any of these categories. Of these algorithms, DME is broken, HiMQ-3 has the flaw in EUP-CMA security proof, Gui has the vulnerability in the parameter set.

The multivariate-based candidates for KEM have the original construction. However, they were broken. Below we briefly present the candidates for KEM.

Table 5.2 contain space requirements for KEM implementations, *sec* denotes the claimed NIST level of security, and *sk* (private key), *pk* (public key), *c* (ciphertext) and $pk + c$ (public keys + ciphertexts), entries all denote length in bytes (according to [137]).

Table 5.3 contains running times for KEM implementations, *sec* denotes the claimed NIST level of security, and `KeyGen` (key generation), `Encap` (encapsulation), and `Decap` (decapsulation) entries all denote milliseconds needed to complete each process (according to [20]).

NPQCC/CFPKM. CFPKM was presented by O. Chakraborty et al. [59]. This scheme is based on the hardness of the PoSSoWN problem. CFPKM is a multivariate quadratic KEM. Two different version of the KEM were presented, CFPKM128 and CFPKM182. We give a simplified description of CFPKM from [176].

Let $q = 2^c$, where $c \geqslant 1$. A public key consists of a seed *seed* and a vector $\mathbf{b}_1 \in \mathbb{Z}_q^m$, where *seed* is expanded into a list of m quadratic polynomials $F(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))$ with small coefficients in n variables $\mathbf{x} = (x_1, \ldots, x_n)$ over $\mathbb{Z}_q$. The secret key is a short vector $\mathbf{sa} \in \mathbb{Z}_q^n$ and the vector $\mathbf{b}_1$ is found as $\mathbf{b}_1 = F(\mathbf{sa}) + \mathbf{e}_1$, where $\mathbf{e}_1$ is a vector of small random errors. To encapsulate, one chooses a random short vector $\mathbf{sb} \in \mathbb{Z}_q^n$. The ciphertext is then $F(\mathbf{sb}) + \mathbf{e}_2$, where $\mathbf{e}_2$ is also a vector of small random errors, in addition to some reconciliation information. The key is obtained as the most significant bits of $\mathbf{b}_1 \odot F(\mathbf{sb})$, where $\odot$ is the component-wise product. The decapsulator obtains the same key by computing $F(\mathbf{sa}) \odot (F(\mathbf{sb}) + \mathbf{e}_2)$ and taking the most significant bits of this vector's components, and by correcting occasional errors when necessary.

An attack breaking the IND-CPA security of both versions of the KEM was presented by R. Steinfeld [174]. This attack efficiently decrypts the private key, given the ciphertext and the public key. Thus, NPQCC/CFPKM is insecure.

NPQCC/DME. DME was presented by I. Luengo et al. [143]. This scheme is based on a new construction of the central maps. DME is composed of both KEM and a signature algorithm. Two different version of the KEM were presented, DME-144 and DME-288.

In DME, the elementary pieces are three linear maps, alternated with matrix exponentiations. The resulting composition is a map with a high degree and a moderate number of monomials.

Let $q = p^e$, where p is a prime and $e \geqslant 1$. Let a matrix $A = (a_{i,j}) \in \mathbb{Z}_q^{n \times n}$. One can define a kind of exponentiation of vectors by using a monomial map $G : \mathbb{F}_q^n \to \mathbb{F}_q^n$ associated to the matrix A as follows:

$$G(x_1, \ldots, x_n) = (\prod_{i=1}^{n} x_i^{a_{1,i}}, \ldots, \prod_{i=1}^{n} x_i^{a_{n,i}}).$$

The public key of DME is a map $\mathcal{P} : \mathbb{F}_q^{nm} \to \mathbb{F}_q^{nm}$ obtained as composition of five maps:

$$\mathcal{P} = L_3 \circ G_2 \circ L_2 \circ G_1 \circ L_1.$$

The components of $\mathcal{P}$ and $\mathcal{P}^{-1}$ are given by polynomials in $\mathbb{F}_q[x_1, \ldots, x_{nm}]$. The maps $L_1 : \mathbb{F}_q^{nm} \to \mathbb{F}_{q^n}^m$, $L_2 : \mathbb{F}_{q^n}^m \to \mathbb{F}_{q^m}^n$ and $L_2 : \mathbb{F}_{q^m}^n \to \mathbb{F}_q^{mn}$ are $\mathbb{F}_q$-linear isomorphism. The maps L_1, L_1 satisfie that for $x, y \in (\mathbb{F}_q^n \setminus \{0\})^m$ values $L_1(x) \in (\mathbb{F}_q^n \setminus \{0\})^m$, $L_2(y) \in (\mathbb{F}_q^m \setminus \{0\})^n$. The maps G_1 and G_2 are monomial maps with the invertible determinant and entries powers of p. The maps G_1 and G_2 are chosen in such a way that each component of $\mathcal{P}$ have few monomials and each component of $\mathcal{P}^{-1}$ has a huge number of monomials.

DME is used for KEM in a standard way but for KEM there is no need to use the padding.

The parameters $m = 3$, $n = 2$, $q = 2^{24}$ (version DME-144) and $m = 3$, $n = 2$, $q = 2^{48}$ (version DME-288) are proposed for 128 and 256 bit security respectively.

The authors of DME have estimated the security against Grobner basis attacks and some other standard attacks. However, in [38] it was shown that DME is insecure.

Table 5.2: Space requirements for the multivariate-based KEM implementations

Implementation	*sec*	*sk* (B)	*pk* (B)	*c* (B)	$pk + c$ (B)
DME-144	1	144	1152	144	1296
DME-288	5	288	2304	288	2592
CFPKM128	1	128	696	729	1425
CFPKM182	3	182	928	729	1657

Table 5.3: Running times for the multivariate-based KEM implementations

Implementation	*sec*	`KeyGen`	`Encap`	`Decap`
DME-144	1	25.79	0.12	0.59
DME-288	5	95.51	0.847	4.19
CFPKM128	1	183	188	176
CFPKM182	3	490	492	502

5.4.5 Conclusion

In general, MB-schemes are fast and requires insignificant computational resources. However, their public keys are quite large and key generation can be slow.

The most active area in the design of MB-schemes is signature. Usually, the multivariate-based signature algorithms are based on MB-schemes which are pretty well studied. On the other hand, research has shown that the development of PKE and KEM based on MB-schemes is difficult.There have been several attempts to build the multivariate-based PKE and KEM. However, most of them, including all candidates of NPQCC, have been broken.

Thus, the main disadvantage of the multivariate-based PKE and KEM is that they are relatively new and their resistance against possible attacks is unclear.

5.5 Other schemes

5.5.1 Introduction

In this section we consider submissions to NPQCC that do not belong to the main platforms LBC, CBC and MBC. Their strength is based on other hard problems. For each of these schemes, we will provide brief descriptions for a scheme and hard problem. At the end of this section, we will compare all these schemes with each other.

5.5.2 Giophantus

Description. Giophantus cryptosystem proposes a post-quantum public key encryption scheme whose security is based on finding small solutions of indeterminate equations, to which approximation attacking algorithms (e.g., LLL and BKZ) cannot be applied directly when the equations are non-linear. This scheme was developed by Toshiba Corporation.

This scheme belongs to algebraic surface cryptosystems (ASC). Such cryptosystems were proposed in 2006 year [6] for the first time. In Giophantus a noise is added to the cipher polynomial to be secure against known attacks. Authors argue that their scheme is provably secure in the sense of IND-CPA under the indeterminate equation of the learning with errors assumption. The Giophantus comes from the Diophantine equations used as the general term for the indeterminate equations. The security of this cryptosystem depends on the computational hardness of solving indeterminate equations. Solving non-linear indeterminate equations is a well-known hard problem in general. In particular, it is known that there is no general solution for equations over $\mathbb{Z}$ (Hilberth's 10th problem) or $\mathbb{F}_q[t]$ and no general algorithm for solving them.

In January 2018, it was proposed an attack in the comments to the submission on the NIST site. But the method to prevent this attack was proposed later.

Hard problem. The algebraic surface over $\mathbb{F}_p$ is the set of solutions of a three-variable equation $X(x, y, t) = 0$ over a field $\mathbb{F}_p$. It is difficult to find parametrized curves (x, y, t) on X that can be written in form $(x, y, t) = (u_x(t), u_y(t), t)$, where $u_x(t)$ and $u_y(t)$ are polynomials in t over $\mathbb{F}_p$. The security of ASC depends on the following problem: if $X(x, y, t) = 0$ is a surface over $\mathbb{F}_p$ then the problem of finding a parametrized curve $(x, y, t) = (u_x(t), u_y(t), t)$ on X is called section finding problem. A section can be considered as a solution of $X(x, y) = 0$, which is indeterminate equation over the ring $\mathbb{F}_p[t]$. This problem can be reduced to solving the system of indeterminate equations [6]. And it is proven [71] that there is no general algorithm to solve this system.

Key sizes. Giophantus provides three parameters sets. They are related to 1st, 2nd and 3rd NIST levels.

For the first level pk size is 14412 bytes, sk – 602 bytes, ct – 28824 bytes. For the second level pk size is 20796 bytes, sk – 868 bytes, ct – 41592 bytes. For the third level pk size is 27204 bytes, sk – 1134 bytes, ct – 54408 bytes.

5.5.3 Mersenne-756839

Description. Mersenne-756839 is KEM cryptosystem that can be seen as belonging to a family that started with the NTRU cryptosystem. The common idea of all these cryptosystems is to work with elements in a ring which are hidden by adding some small noise. This notion of smallness needs to be somewhat preserved under the arithmetic operations.

First proposal of Mersenne [3] only allowed to encrypt a single bit at a time. But Mersenne-756839 permits to encrypt many bits at a time.

Hard problem. The security of Mersenne-756839 is based on the Mersenne Low Hamming Combination Assumption [4]. This assumption can be rewritten as follows: for a given uniformly random n-bit string R, when it is considered that $T = F \cdot R + G(\mathrm{mod}p)$, where the binary representation of F and G modulo p has low Hamming weight, then T appears pseudorandom, i.e., it seems hard to obtain any non-trivial information about F,G from R, T.

Key sizes. Mersenne-756839 provides only one parameter set for the third NIST level. For this level parameters are the following: pk size is 184800 bytes, sk – 22000 bytes, ct – 156400 bytes.

5.5.4 SIKE

Description. Supersingular isogeny key encapsulation (SIKE) is a protocol that is based on a key-exchange construction Supersingular Isogeny Diffie-Hellman (SIDH) that was firstly proposed in [122]. This protocol is deeply researched by Microsoft Corporation. SIKE uses construction of elliptic curves over finite fields. SIKE proposes both KEM and PKE schemes.

Hard problem. The main termin in SIDH is an isogeny. Isogeny is non-constant rational map that map one elliptic curve to other. The security of SIKE relies on the supersingular isogeny walk problem: given two elliptic curves E, E' in the same isogeny class, find a path made of isogenies of small degree between E and E'. The best algorithm against this problem is a meet-in-the-middle strategy that, in average, requires a number of elementary steps proportional to the square root of size of the isogeny class of E and E'. But this attack is not simply applied on SIKE. Another attack is very efficient polynomial-time attack against SIDH with static keys [91]. This attack can be applied as CCA only on PKE scheme but not to KEM. As a result of security analysis authors prove that a KEM scheme is IND-CCA secure and PKE scheme is IND-CPA secure.

Key sizes. SIKE proposes three parameter sets `SIKEp503`, `SIKEp751` and `SIKEp964`. For `SIKEp503` pk size is 378 bytes, sk – 434 bytes, ct – 402 bytes. For `SIKEp751` pk size is 564 bytes, sk – 644 bytes, ct – 596 bytes. For `SIKEp964` pk size is 726 bytes, sk – 826, ct – 766 bytes. As we can see SIKE has the small key and ciphertext sizes in all parameter sets.

5.5.5 Ramstake

Description. Ramstake is a KEM cryptosystem that relies on a relatively new and understudied hard problem. This scheme is based on a noisy Diffie-Hellman protocol. Since this hard problem (also as a scheme) is young it needs a few years attention of cryptographic community on this problem to use it in the real projects.

Hard problem. Ramstake relies on the hardness of the same problem as Mersenne-756839. It is the Low Hamming Combination Assumption [4]. This problem was described above.

Key sizes. Ramstake provides two parameter sets: `Ramstake RS 216091` and `Ramstake RS 756839`. For `Ramstake RS 216091` pk size is 27044 bytes, sk – 54056 bytes, ct – 28064 bytes. For `Ramstake RS 756839` pk size is 94637 bytes, sk – 189242 bytes, ct – 96111 bytes.

5.5.6 Post-quantum RSA Encryption

Description. Post-quantum RSA (pqRSA) encryption is the extension of famous modern cryptosystem RSA. pqRSA uses extremely large RSA keys to stop Shor's algorithm. It uses many relatively small secret primes, and small encryption/verification exponents, so that computations with such large keys are affordable for the legitimate users. pqRSA provides a KEM and a PKE schemes. Also it provides signature algorithm.

Hard problem. The hard problem for pqRSA is the same as for ordinary RSA: integer factorization. The new version of RSA designed to be strength against main known attacks: factorization using Shor's quantum algorithm (in post-quantum epoch it works faster than the number-field sieve) and the elliptic curve method.

Key sizes. pqRSA proposes four parameter sets: `pqrsa15`, `pqrsa20`, `pqrsa25` and `pqrsa30`. Public keys, private keys and ciphertexts have the same values for each set and they are 2^{15}, 2^{20}, 2^{25} and 2^{30} bytes respectively. The PKE ciphertexts have the same size as keys if the transmitted messages are short enough.

5.5.7 Guess Again

Description. Guess Again cryptosystem proposes a post-quantum public-key encryption scheme whose security is based on the probability of adversary to guess the secret bit. In existing popular encryption schemes adversary competes with receiver. In Guess Again the sender tries to guess the reciever's decryption key, but the adversary tries to guess the sender's secret bit. Therefore they may have different probability spaces for their guessing and sender may have advantage over adversary. This scheme is scheme with possible decryption errors. Probability of of decryption errors is minimized by encryption one bit thousands times. It makes this scheme inapplicable in real life. On the other hand it stays secure even if passive adversary has computationally unbounded resources.

Hard problem. Bob/Alice randomly chooses private key $b/a \in [0, n-1]$ and calculate public key $B = F(b)/A = F(a)$, where $n \in N$ and F — random walk function. To encrypt message Alice tries to guess b knowing B. For Mallory probability to guess b is the same as for Alice. But Mallory wants to know not b, what is the secret bit that transmit Alice. It means that Malory's goal is to guess what Alice thinks b is. This reasoning leads to the fact that Alice and Mallory have different probability spaces for their goals. Alice makes her probability to guess b greater than $1/2$ with help of random walk process. And namely by the following fact: if Alice and Bob do independent random walks starting at two random points, a and b, respectively, then the conditional probability $\mathbf{P}\{b < a \mid A < B < a\}$ is higher when the number of steps in Alice's random walk is larger (with the number of steps in Bob's random walk fixed). Authors of Guess Again propose parameters that leads to Alice guessing probability equals 0.55. Maloory's probability stays $1/2$. Increasing chiphertext length for one bit we can increase receiver's probability to correctly decrypt transmitting bit.

Key sizes. Authors propose parameters that promise probability 0.999996 to correctly decrypt one bit with the following sizes: sk — 18000 bits, pk — 16000 bits, ct for 1 bit —- 18000 bits. Notice that the numbers of encrypted bits do not affect on sk and pk sizes.

5.5.8 Withdrawn schemes

HK17 and RVB schemes were withdrawn due to some important notes that were provided in comments on the NIST site. Therefore there is no need to explore these schemes deeply.

5.5.9 Comparison

Time. There are three main algorithms in the PKE schemes for which time comparison must be done. They are `Gen`, `Encr` and `Decr`. For the KEM schemes, they are `Gen`, `Encap` and `Decap`. All these algorithms are described in Section 5.1. In Table 5.4 we provide time measurements for all valid candidates (that are not withdrawn) from this section. This table includes data for all NPQCC security levels (see 1.3.1). All data are taken from [20]. In the table, `Encr` and `Decr` also mean `Encap` and `Decap` for the KEM schemes.

Table 5.4: Time measurements

Scheme	Level 1			Level 2			Level 3		
	`Gen`	`Encr`	`Decr`	`Gen`	`Encr`	`Decr`	`Gen`	`Encr`	`Decr`
Giophantus	12.14	22.01	41.31	22.03	49.88	94.37	32.16	78.95	151.34
Mersenne-756839	-	-	-	-	-	-	6.02	9.23	18.18
SIKE	26.4	43.22	46.11	85.99	140.98	151.85	-	-	-
Ramstake	2.35	4.34	8.92	10.88	19.82	38.46	-	-	-
pqRSA (Level 2)	-	-	-	1336	8.39	46.99	-	-	-
Guess Again	-	-	-	-	-	-	38.7	2634	1.38

As we can see from Table 5.4, the best solution (in time measurements) is Mersenne-756839.

Memory. There are three main entities for which memory costs comparison must be done. They are pk, sk and ct. As well as for the time measurements in Table 5.5 we provide data for all valid candidates for various security levels. All sizes are in bytes.

Table 5.5: Memory costs

Scheme	Level 1			Level 2			Level 3		
	sk	pk	ct	sk	pk	ct	sk	pk	ct
Giophantus	600.5	14412	28824	8665.5	20796	41532	1133.5	27204	54408
Mersenne-756839	-	-	-	-	-	-	22000	184800	156400
SIKE	434	378	402	644	564	596	826	726	766
Ramstake	54056	27044	28064	189242	94637	96111	-	-	-
RSA (Level 2)	-	-	-	2^{15}	2^{15}	2^{15}	-	-	-
Guess Again	-	-	-	-	-	-	2000	2250	$32 \cdot 10^4$

As we can see from Table 5.5, the best solution (in memory costs) is SIKE.

5.5.10 Conclusion

Let us compare all PKE/KEM algorithms in Other category and try to choose the best solution from these cryptosystems. First of all two cryptosystems (RVB and HK17) were withdrawn because some defects and mistakes were detected in their algorithms. Algorithm Guess Again has incredible large ciphertext (to encrypt 1 bit of plaintext we need 18000 ciphertext bits). The pqRSA has a large key generation time and too big key sizes. The last algorithms seem to be the best. But they have disadvantages too. Giophantus, Mersenne-756839 and Ramstake cryptosystems have large public key and ciphertext sizes. For these cryptosystems also there were no proposed signature algorithms in NPQCC. Besides, these cryptosystems are relatively young.

The last remaining scheme is SIKE. SIKE has the smallest key sizes among all submissions from Other category. But its key generation, encryption and decryption algorithms are relatively slow. Another potential drawback is that SIKE is relatively young cryptosystem. We

should also note that the IBC (Isogeny-Based Crypto) platform to which SIKE belongs does not present any DS submission to NPQCC.

5.6 The final scheme

5.6.1 Decisions

Let us we explain our decisions regarding the final PKE algorithms. We present the explanations in the form of questions at the encountered "crossroads" and corresponding answers. The main "crossroads" are presented in Figure 5.1.

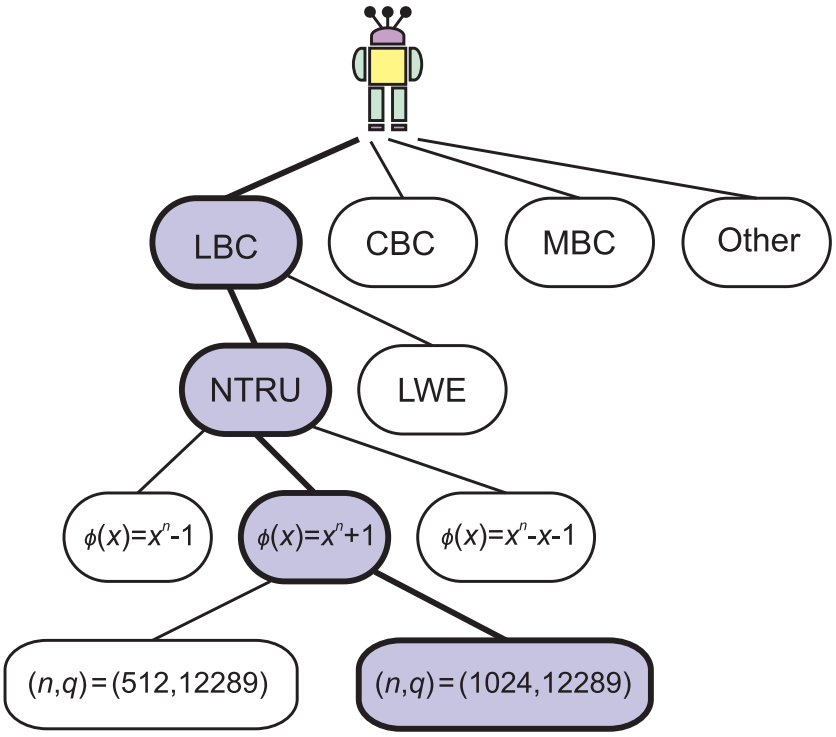

Figure 5.1: A decision tree

5.6.1.1 Why LBC?

We choose the LBC platform because it meets all the specified requirements (see 5.2.10) and because these requirements are not met by any other platform.[16] More precisely, let us answer three subordinate questions.

Why not CBC? CBC is a well-proven and stable platform. The platform appeared much earlier before LBC in the form of the McElice PKE system (1978). CBC provides sufficiently high performance, adapts well to the ternary logic. The platform contains DS schemes although they are not so convenient as PKE/KEM counterparts (just as in the LBC case).

[16] It is significant that LBC cryptosystems were used in both Google's experiments devoted to the integration of post-quantum cryptography with TLS 1.3. See `https://www.imperialviolet.org/2016/11/28/cecpq1.html` and `https://www.imperialviolet.org/2018/12/12/cecpq2.html`.

The main drawback of CBC is very large public keys. Among all CBC submissions to NPQCC, only LAKE and LOCKER offer competitive lengths of keys.

These two submissions are designed by the same team, the submissions differ only in details. Since LAKE and LOCKER meet all the requirements, we analyzed them more thoroughly.

The authors of LAKE/LOCKER postulate that the main difference between CBC and LBC is essentially a choice of a metric: the Hamming metric in CBC vs Euclidian in LBC. Focusing on the metric, the authors propose the third alternative: the rank metric introduced by Gabidulin in 1985. Interestingly, LAKE and LOCKER use NTRU-like long-term parameters, that is why public keys are so compact. Coordinates of a noise vector which is used during encryption are selected from a subspace of $\mathbb{F}_q^n$ of a small dimension. It means that the noise vector has a small rank norm.[17] The rank norm is so small that Alice can decode a noisy ciphertext and therefore decrypt it. Unfortunately, Alice should use a rather complicated iterative algorithm. Because of the iterations, the decryption time can differ depending on a private key. This creates the prerequisites for so-called timing attacks.

A drawback of LAKE is that a probability of decryption failures can be quite large. It is guaranteed only that the probability does not exceed 2^{-32}. LOCKER decrease this threshold to 2^{-64}. Unfortunately, according to independent estimates [20], LOCKER is approximately 4 times slower than LAKE. And LAKE's speed is at the same level as NPQCC/NTRUEncrypt's.

In whole, LAKE / LOCKER looks good ideologically although the motivation of the rank metric is somewhat unclear. No attacks are known, but the design is rather recent. So there are no clear advantages over NTRU. It makes no sense to choose the LAKE / LOCKER subplatform in such situation.

Why not MBC? MBC is a highly unstable platform accompanied by a negative history of successful attacks. The evolution of the platform is very similar to the evolution of knapsack cryptography where inventors of cryptosystems were permanently forced to patch them. At the moment, all MBC submissions to NPQCC have been compromised.

Another disadvantage of MBC is very large public keys. This drawback is overcome only in NPQCC/DME, but this submission was successfully attacked first.

Why not Other? The remaining platforms are mostly experimental. They need to become stable and acquire credit confidence. It makes no sense to use them right now.

The only exception is the IBC (Isogeny-Based Cryptography) platform. It is represented by the NPQCC/SIKE submission. A great advantage of SIKE is compact public keys and ciphertexts. The main disadvantage is high computational complexity (according to [20], it is 20-40 times slower than NPQCC/NTRUEncrypt).

5.6.1.2 Decryption Failures?

In many settings, decryption failures are acceptable if they occur with only small probabilities. For example, if a TLS client uses PKE to transfer a session key and if the interacting server cannot decrypt this key, then entities just try again. But we are in the MAM2 settings and here we should avoid decryption failures as much as possible. Indeed, senders should have strong guarantees that their keyloads will be decrypted even in the distant future.

Another point is that standard ProvSec notions, which are related to security of PKE/KEM

[17] For comparison, LBC usually use noise vectors with a small number of zero coordinates (that is, with small Hamming norm).

systems (for example, IND-CCA, see Section 5.1), work without decryption failures. Of course, we can modify these notions but it would be better if we fit the standard ones.

From the practical point of view, failures, which probabilities are below 2^{-256}, are almost for never. We can ignore them.

5.6.1.3 PKE or KEM?

PKE is better. MAM2 needs PKE, not KEM, for sending the same session key to different recipients. We can convert KEM into PKE using hybrid encryption (see Section 5.1). But it would be better to use PKE directly.

5.6.1.4 NTRU or LWE?

This is probably the most difficult question.

NTRU and LWE are two branches of LBC (see 5.2.6, 5.2.7). The LWE cryptography is "on the march". This can be seen from its wide representation in NPQCC. Almost every year of the last decade brings new breakthroughs in the LWE field. LWE is better justified, there are more known reductions of thoroughly-tested hard lattice problems to the problems related to the security of LWE schemes. The similar reductions for NTRU are also known (see pNE in 5.2.6), but they require noncompetitive large q.

A direct competitor of NTRU in LWE is the RLWE subbranch. The inherent difference between NTRU and RLWE is that RLWE uses random lattices generated by secret polynomials s while NTRU lattices are more structured, they are generated by public polynomials h which have a special internal form. Having a structure is, on the one hand, a potential weakness. But 20-year analysis of NTRU does not reveal serious weaknesses that cannot be overcome. On the other hand, the presence of a structure is the possibility of building the simplest and therefore most efficient solutions, which is what actually happens in the case of NTRU.

With our current level of understanding, we consider the choice between NTRU and LWE as a matter of taste and intuition. We were inclined to NTRU for the following reasons:

1. A clever cryptographic idea.
2. A long-term approbation (20 years without serious attacks).
3. Perfectly fit the ternary logic (plaintexts and private keys are usually strings of trits).
4. A rather broad representation in NPQCC: 3 encryption systems (NTRUEncrypt, NTRUPrime, NTRU-HRSS-KEM); 2 signature systems (pqNTRUSign, Falcon); and even 2 encryption systems from the CBC class (LAKE, LOCKER) exploit the idea of NTRU.
5. Ability to build DS algorithms. There are prerequisites for using the same long-term parameters both in signature and encryption algorithms.
6. Keys and ciphertexts are about 2 times shorter than in RLWE (without reconciliation).

5.6.1.5 Why not NTRU Classic?

NTRU Classic is the main subbranch of NTRU maintained by the NTRU founders themselves. NTRU Classic is under international standardization. NTRU Classic is represented in

NPQCC with two submissions: NTRUEcnrypt and pqNTRUSign. We consider NTRUEncrypt as one of the favorites of NPQCC.

But we do not use NTRU Classic in MAM2. The reasons are:

1. Classic long-term parameters seem to be not very effective. We want to use modern alternatives which provide better performance and very compact implementations.

2. We want to suppress decryption failures (NTRUEncrypt guarantees that probabilities of decryption failures are below 2^{-196} for the first paramset and 2^{-112} for the second one).

3. We want to abandon restrictions on numbers of 0's, 1's and -1's in the ternary polynomials f and g and generate them uniformly, that is, in the easiest way.

4. If a secret polymial g is not coprime with ϕ then so is the public polynomial h. In other words, h can leak information about g. The problem may be serious with some modifications of the classic NTRU polynomial $\phi(x) = x^n - 1$. We want to make g coprime with ϕ.

5. We want to replace the somewhat heavy NAEP with a simpler alternative preferably built using AE (authenticated encryption) algorithms already presented in MAM2.

5.6.1.6 Why not pNE?

pNE is a ProvSec extension of NTRU Classic. It is included in NPQCC/NTRUEncrypt in the form of the 3rd paramset.

Unfortunately, increased security guarantees of pNE come together with inefficiency: According to official data from the NTRUEcnrypt submission, NTRU algorithms of the 2nd paramset are 5-10 times faster than of the 3rd one.

5.6.1.7 Why $\phi(x) = x^n + 1$?

We use the modular polynomial $\phi(x) = x^n+1$ because it nicely fits NTT. Other alternatives are listed and discussed in 5.2.5.

We are attentive to the arguments of the authors of NPQCC/NTRUPrime about "worrisome algebraic structures" induced by the chosen $\phi(x)$. But we hope that these fears remain only fears. We lean to the opinion of the NPQCC/NTRU-HRSS team who say that "*[the authors of NTRUPrime] pay a very large penalty for requiring a large Galois group (NTT-friendly polynomials always have small Galois groups) and an inert modulus (NTT-friendly primes are never inert).*"

5.6.1.8 Why $(n, q) = (1024, 12289)$?

To be completely NTT-friendly, the following conditions must be satisfied: n is a power of 2, q is prime, q is congruent to 1 mod $2n$.

We have two combinations of appropriate (n, q) with small q and n from the safe range $[500, 1500]$. They are $(512, 12289)$ and $(1024, 12289)$. The latter option is preferable since it provides a more adequate level of security: $n = 512$ is too risky.

We should mention that the rings $Z_{12289}[x]/(x^{512}+1)$ and $Z_{12289}[x]/(x^{1024}+1)$ are very popular in LBC, especially in RLWE (see Table 5.1). Besides effectiveness, there exist security reasons in favor of these rings (see 5.2.7).

It is important, that $q = 12289$ is close enough to $27^3 = 19683$. So it is very convenient to store each element of $\mathbb{Z}_q$ as a trint (3 trytes). We need $n = 1024$ trints or 9216 trits to store a polynomial $u(x) \in \mathbb{Z}_q[x]/(\phi(x))$ which can stand as a public key or ciphertext.

In fact, we do not have much if we want to use NTT-friendly (n, q). One "mysterous" thing about the chosen (n, q) is that $1 + 6n \approx q/2$. From the NTRU perspective, if we want to avoid decryption failures, then we should satisfy the inequality $1 + 6n < q/2$. And we almost satisfy: $1 + 6n = 6145$, $q/2 = 6144.5$. A tiny overlap yields a tiny probability of decryption failures: 1 failure in 2^{4444} experiments (see comments on Theorem 5.1). It is beyond any reasonable thresholds.

Let us compare our parameters (n, q) with close parameters from NTRU submissions to NPQCC. NTRUPrime uses $n = 743$ and the very small $q = 2048$. Due to the smallness, decryption failures can occur with a rather large probability. To avoid failures, NPQCC/NTRUPrime uses $(n, q) = (761, 4591)$ and NPQCC/NTRU-HRSS uses $(n, q) = (701, 8192)$.

We see that our (n, q) are heavier. But the difference isn't very large from the practical point of view: to store elements of $\mathbb{Z}_{4591}$ or $\mathbb{Z}_{8192}$ it is natural to use 8 or even 9 trits ($3^7 < 4591 < 3^8 < 8192 < 3^9$). So the total length of a public key is either $761 \cdot 8 = 6088$ or $701 \cdot 9 = 6309$ trits, not so dramatically smaller comparing with our choice.

An additional argument in favor of relatively large (n, q) is that DS algorithms usually use heavier parameters than PKE ones. It may well be that with the chosen parameters we will be able to launch DS.

Observe that the polynomial $\phi(x) = x^n + 1$ splits over $\mathbb{Z}_q$ into the product of linear polynomials. More precisely, all roots of $\phi(x)$ lay in $\mathbb{Z}_q$. The smallest roots in magnitude are $x = \pm 7$. Other roots: $\pm 8, \pm 20, \pm 24, \pm 50, \ldots$.

Potentially, if r is a some root, then $c(r)$ reveals information about $m(r)$. To obtain such information an adversary has to determine $f(r)$, $g(r)$ from the congruence $h(r)f(r) \equiv g(r) \bmod q$. In the case of NTRU Classic, it is an easy problem because $\phi(x) = x^n - 1$ has a small root $r = 1$ and $f(r), g(r)$ will be small. But in our case $|r| \geqslant 7$ and the adversary cannot exploit the smallness of $f(r), g(r)$.

5.6.2 Algorithms

Now we specify final PKE algorithms and comment on them. The algorithms are presented in a laconic form convenient for analysis. Detailed specifications are provided outside this report.

We use standard notations introduced in Section 5.2: $R = \mathbb{Z}[x]/(x^n+1)$, $R_3 = \mathbb{Z}_3[x]/(x^n+1)$, $R_q = \mathbb{Z}_q[x]/(x^n+1)$. An element of R_q can be lifted to R and then reduced mods 3 to get into R_3. In other direction, an element of R_3 can be lifted to R and then interpreted as an element of R_q. To indicate one of the interpretations, we write "mods 3" or "mods q".

Let us recall that $(n, q) = (1024, 12289)$.

Algorithm NTRU.Gen

Input: $\perp$.

Output: $f \in R_3$ (a private key), $h \in R_q$ (a public key).
Steps:

1. Generate $f, g \xleftarrow{R} R_3$.
2. Test if the polynomials $1 + 3f$ and g are invertible mods q. If not, then go to Step 1.
3. $h \leftarrow 3g/(1 + 3f) \bmod q$.
4. Return (f, h).

Usually in NTRU, a private key contains both f and g. But g is not used for decryption, so we exclude it from key material. In any case, g can be easily reconstructed by f and h.

Another important point is that f and g are chosen uniformly. It is the most convenient way of generation of secret parameters in cryptography. It can be easily implemented using conventional generators of random or pseudorandom numbers. For comparison, in NTRU Classic, f and g are polynomials with prescribed numbers of 0's and ± 1's.

To test invertibility of a polynomial $u \in R_q$, one can apply NTT and check if $\hat{u} = \mathrm{NTT}(u)$ does not contain zero coefficients. Since there are $(q-1)^n$ appropriate $\hat{u}$ and NTT sets a bijection on R_q, the number of invertible elements of R_q is $(q-1)^n$. In other words, the probability of u to being invertible is

$$\left(1 - \frac{1}{q}\right)^n \approx e^{-n/q}.$$

It is approximately 0.92 with our (n, q).

Note that we test invertibility not general polynomials of R_q, but extremely small polynomials g and $1 + 3f$. It is usually assumed that these small polynomials are invertible mods q with the same probability $\approx e^{-n/q}$. It yields that Step 2 would be succesful with probability $\approx e^{-2n/q} \approx 0.84$.

Unfortunately, the reasonings about invertibility of small polynomials are only hypothetical. To make generation aspects rigorous, in [18] NTRU Classic is modified in such a way that the polynomials involved are invertible with fully controllable probabilities. A drawback is that amount of key material, as well as complexities of resulting algorithms, increase approximately twice. We think that expenses are too high.

Note that we insist that g is invertible mods q. For comparison, in NTRU Classic only the polynomial $1 + 3f$ (or f in earlier versions) must be invertible.

We make g invertible for 2 reasons:

1. Invertibility of g facilitates a proof of security (see below).
2. If g is not invertible, then h is not invertible too. In this case, h reveals common factors of g and $x^n + 1$. Let us recall that $x^n + 1$ splits over $\mathbb{Z}_q$, so these factors can only be linear. Each linear factor $x - \alpha$ yields the congruence $g(\alpha) \equiv 0 \bmod q$ which decreases uncertainty about g in approximately q times.

To move further, we have to introduce auxilliary cryptographic mechanism — an AE (authenticated encryption, see Section 2) cryptosystem which provides 2 algorithms: $\mathcal{W}$ (Wrap)

and $\mathcal{U}$ (Unwrap). The first algorithm takes a key k and a plaintext pt and returns a ciphertext ct. It acts injectively in pt for fixed k: $\mathcal{W}(k, pt) \neq \mathcal{W}(k, pt')$ if $pt \neq pt'$. The second algorithm takes (k, ct) and returns either pt or $\perp$:

$$\mathcal{U}(k, ct) = \begin{cases} pt, & \text{if there exists } pt \text{ such that } \mathcal{W}(k, pt) = ct, \\ \perp, & \text{otherwise.} \end{cases}$$

An ideal AE system is when $\mathcal{W}$ is chosen uniformly at random from the set of all mappings $(k, pt) \mapsto ct$ that are injective in pt for fixed k and when $\mathcal{U}$ inverts $\mathcal{W}$ where possible. In the next paragraph, we will show how to model such an ideal system.

We suppose that AE keys are from R_q, a plaintext pt is from $\mathbf{T}^l$ where $l = 243$ (actually, pt is a session key of the predetermined length) and ct is a polynomial from $R_3 \sim \mathbf{T}^n$. In Specification, we instantiate $\mathcal{W}$ and $\mathcal{U}$ using constructions from Section 2:

1) k is absorbed as a key;

2) pt is absorbed and encrypted;

3) a MAC of $n - l$ trits is squeezed;

4) a ciphertext ct is a concatenation of the encrypted pt and the MAC.

In our instantiation, conversions between polynomials and ternary words are natural and transparent. Note that since $l = 243$ is (significantly) less than $n = 1024$, pt is (significantly) expanded during wrapping.

Algorithm NTRU.Encr

Input: $m \in \mathbf{T}^l$ (a plaintext), $h \in R_q$ (a public key).
Output: $c \in R_q$ (a ciphertext).
Steps:

1. $r \xleftarrow{R} R_3$.
2. $s \leftarrow h \cdot r \bmod q$.
3. $t \leftarrow \mathcal{W}(s, m)$.
4. $c \leftarrow (s + t) \bmod q$.
5. Return c.

Algorithm NTRU.Decr

Input: $c \in R_q$ (a ciphertext), $f \in R_3$ (a private key).
Output: $m \in \mathbf{T}^l$ (a plaintext) or $\perp$ (a error).
Steps:

1. $t \leftarrow c(1 + 3f) \bmod q \bmod 3$.
2. $s \leftarrow (c - t) \bmod q$.

3. Return $\mathcal{U}(s,t)$.

In the algorithms, we follow the general line of NTRU Classic. The main difference is that we accompany internal NTRU Problem not with NAEP (see 5.2.6) but with AE. Note that NAEP of NPQCC/NTRUEncrypt requires 2 hash calls during encryption / decryption and one additional multiplication over R_q during decryption. The last multiplication is used to check decrypted data. For comparison, our scheme requires only 1 AE-call during encryption / decryption. The soundness of decrypted data is verified through AE. For clarity, in the instantiation of the scheme, we use an additional PRNG-call to generate r during encryption, that is, to implement the step $r \xleftarrow{R} R_3$. Similar functionality, the generation of r, is achieved in NPQCC/NTRUEncrypt through a hash call.

Another difference from NTRU Classic is that decryption failures are possible only in a very specific situation.

Theorem 5.1. *Let* $\phi(x) = x^n + 1$, $(n,q) = (1024, 12289)$ *and* $(f,h) \leftarrow$ *NTRU.Gen*$(\perp)$. *A decryption failure*

$$\textit{NTRU.Decr}(\textit{NTRU.Encr}(m,h),f) \neq m$$

occurs for some $m \in \boldsymbol{T}^l$ *only if*

1) the ternary polynomials f *and* $g = h(1+3f)$ *do not contain zero coefficients;*

2) (r,t)*, the ternary polynomials of* *NTRU.Encr*, *take unique values determined by* (f,g).

Proof. Due to the choice of $\phi(x)$, the ith coefficient of $3rg + t(1+3f)$ has the form:

$$t_i + 3\sum_{j=0}^{n-1} \gamma_{ij}(r_j g_{i-j} + t_j f_{i-j}).$$

Here γ_{ij} are either 1 or -1, subtractions in indices are made modulo n. Since all the terms $t_i, r_j, g_{i-j}, \ldots$ are from $\{\pm 1, 0\}$, the resulting coefficient doesn't exceed $1+6n$ in magnitude and, therefore,

$$\|3rg + t(1+3f)\|_\infty \leqslant 1 + 6n.$$

The upper bound is reached if and only if

$$\gamma_{ij} r_j g_{i-j} = \gamma_{ij} t_j f_{i-j} = t_i \neq 0$$

for all $j = 0, 1, \ldots, n-1$. It means that f, g, r, t do not contain zero coefficients and, moreover, (f,g) are uniquely determined by (r,t) and vice versa.

If the upper bound is not reached, then $\|3rg + t(1+3f)\|_\infty \leqslant 6n = 6144$ which is less than $q/2 = 6145.5$. The inequality $\|3rg + t(1+3f)\|_\infty < q/2$ guarantees no decryption failures. □

Let

$$\begin{aligned}
\mathcal{E}_1 &= \{f \text{ does not contain zero coefficients}\},\\
\mathcal{E}_2 &= \{g \text{ does not contain zero coefficients}\},\\
\mathcal{E}_3 &= \{r \text{ takes a prescribed value}\},\\
\mathcal{E}_4 &= \{t \text{ takes a prescribed value}\}.
\end{aligned}$$

Assuming that invertibility of random ternary polynomials does not depend significantly on the number of zero coefficents and considering the used AE system $(\mathcal{W}, \mathcal{U})$ almost ideal (in particular, t doesn't depend statisitically on f, g, r), we get:

$$\mathsf{P}\{\text{decryption failure}\} \approx \mathsf{P}\{\mathcal{E}_1\}\mathsf{P}\{\mathcal{E}_2\}\mathsf{P}\{\mathcal{E}_3\}\mathsf{P}\{\mathcal{E}_4\} \approx \left(\frac{2}{3}\right)^n \left(\frac{2}{3}\right)^n \left(\frac{1}{3}\right)^n \left(\frac{1}{3}\right)^n \approx 2^{-4444}.$$

This probability is below any practical threshold. Without loss of a drop of common sense, we can count that there are no decryption failures.

5.6.3 IND-CCA security

An ideal AE system $(\mathcal{W}, \mathcal{U})$ is chosen at random from the set of all suitable systems. This random selection can be described dynamically, by a sequence of observations, that is, inputs and outputs of the algorithms $\mathcal{W}$ and $\mathcal{U}$. The algorithms are interpreted as oracles, their inputs — as queries, outputs — as answers. Oracles are controlled by a *simulator* S.

For keys k that appeared in the queries, the simulator maintains two lists: W_k and $\perp_k$. The first list contains pairs (pt, ct) such that $ct = \mathcal{W}(k, pt)$ or $pt = \mathcal{U}(k, ct)$. The second list contains ciphertexts ct such that $\mathcal{U}(k, ct) = \perp$. Initially, the lists are empty or even not created.

The lists are updated in the following way:

1. Processing a query (k, pt) to $\mathcal{W}$, the simulator looks for a pair (pt, ct), which first element is prescribed, in W_k.
   (a) If the pair is found, the simulator returns ct.
   (b) If the pair is not found, then S chooses ct at random among ciphertexts that do not appear in either W_k or $\perp_k$. The simulator appends (pt, ct) to the list W_k and again returns ct.
2. Processing a query (k, t) to $\mathcal{U}$, the simulator looks for a pair (pt, ct), which second element is prescribed, in W_k.
   (a) If the pair is found, the simulator returns pt.
   (b) If the pair is not found, then with the probability
   $$1 - \frac{3^l - |W_k|}{3^n - |W_k| - |\perp_k|}$$
   the simulator returns $\perp$ and appends ct to the list $\perp_k$.
   (c) If $\perp$ is not returned, then S chooses pt at random among plaintexts that do not appear in W_k. The simulator appends (pt, ct) to W_k and returns pt.

It is clear that the simulator really models an ideal AE system. Simulation expenses are relatively small: if β is a total number of queries to the oracles, then it is sufficient to spend $O(\beta^2)$ operations (even with the simplest linear search in the lists W_k, $\perp_k$).

Hash functions are often simulated in a similar way and such simulation is usually attributed as ROM (Random Oracle Model). In our case, the simulation somewhat more complicated: it is necessary to maintain the partial injectivity of $\mathcal{W}$ and the partial inverse relation between $\mathcal{W}$ and $\mathcal{U}$. But following the tradition, we also position our simulation as ROM.

Now we are ready to formulate the main result about the security of the proposed PKE algorithms. The result is of a qualitative nature. Quantitative estimates rather cumbersome and, in our opinion, are not of great importance in this specific case. In any case, quantitative estimates can be easily extracted from the proof of the following theorem.

Theorem 5.2. *If NTRU Problem is hard and the PKE system* (`NTRU.Gen`, `NTRU.Encr`, `NTRU.Decr`) *uses an ideal AE system* $(\mathcal{W}, \mathcal{U})$ *generated in ROM, then this PKE system is IND-CCA secure.*

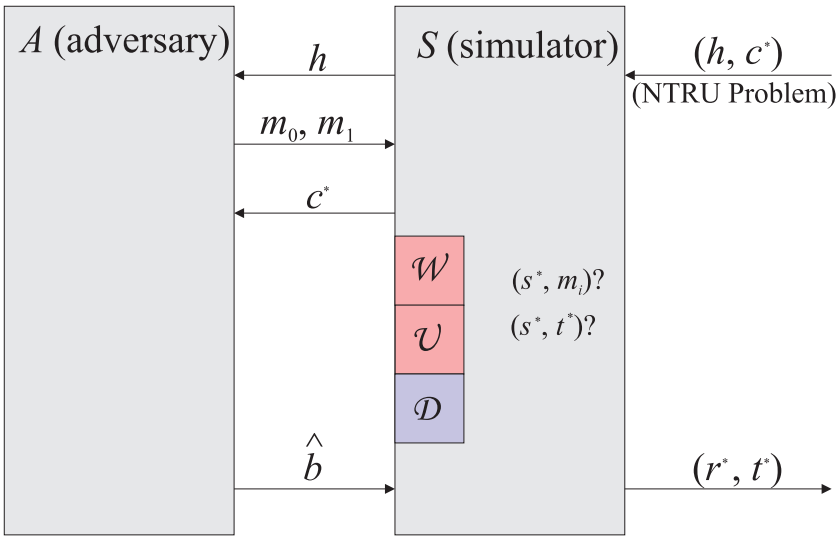

Figure 5.2: IND-CCA settings

Proof. Let us recall the process of solving the IND-CCA problem (see Section 5.1 and Figure 5.2). The simulator generates keys $(f, h) \leftarrow \texttt{NTRU.Gen}(\perp)$ and passes a public key h to an adversary A. The adversary does some calculations and returns to S two different plaintexts: m_0 and m_1. The simulator chooses $b \xleftarrow{R} \{0, 1\}$, calculates $c^* \leftarrow \texttt{NTRU.Encr}(h, m_b)$ and passes c^* to A. The adversary again conducts calculations and returns an estimate $\hat{b}$ of the bit b.

The quality of the adversary's attack is characterized by the advantage

$$\mathbf{Adv}^{\text{IND-CCA}}(A) = \left| \mathsf{P}\{\hat{b} = b\} - 1/2 \right|.$$

A large advantage means that the adversary can distinguish which plaintext corresponds to a given ciphertext and, therefore, the target PKE system is not secure.

During the attack, A can run the algorithm `NTRU.Encr` with any input data (they are public) and can additionally interact with the decryption oracle $\mathcal{D}$, which takes ciphertexts c and returns $\texttt{NTRU.Decr}(c, f)$. The query $c = c^*$ to $\mathcal{D}$ is not allowed. The algorithms `NTRU.Encr` and `NTRU.Decr`, in turn, interact with the simulated oracles $\mathcal{W}$ and $\mathcal{U}$. Direct adversary's queries to these oracles are also allowed.

To prove the theorem, it is required to show that the simulator can use an adversary with a significant advantage for solving NTRU Problem. This means that the simulator receives a public key h and a ciphertext c^*, for which it has to find ternary polynomials r^*, t^* such that

$$c^* = (hr^* + t^*) \bmod q.$$

The simulator tries to solve the problem using A as its subroutine. It is important that S does not know f, otherwise, the problem is solved trivially. And since S does not know f, it needs some efforts for enough accurate simulation of CCA (Chosen Ciphertext Attack) behind A.

The simulator passes h to the adversary and provides for her access to the ROM system $(\mathcal{W}, \mathcal{U})$. Let us further use the `NTRU.Encr` notations. So we replace k by s, pt by m and ct by t. Let $\beta_{\mathcal{W}}$, $\beta_{\mathcal{U}}$ be total numbers of queries to the oracles $\mathcal{W}$ and $\mathcal{U}$ respectively.

The simulator slightly simplifies the ROM rules. First, at Step 1b it chooses t at random from the entire set $\mathbf{T}^n$. It can yield a violation of injectivity of $\mathcal{W}(s, m)$ by m. Fortunately, the probability of non-injectivity is very small, it is at most $\beta_{\mathcal{W}}^2/(2 \cdot 3^n)$. Second, at Step 2c the simulator again returns $\perp$. The new behavior corresponds to the original one if Step2c never occurs. The probability of this event is

$$\prod_{i=0}^{\beta_{\mathcal{U}}-1} \left(1 - \frac{3^l}{3^n - i}\right) > \left(1 - \frac{3^l}{3^n - \beta_{\mathcal{U}}}\right)^{\beta_{\mathcal{U}}} > 1 - \frac{3^l \beta_{\mathcal{U}}}{3^n - \beta_{\mathcal{U}}}.$$

Now the lists $\perp_s$ are redundand and S does not maintain them.

In whole, the probability of distinguishing the simplified simulation of $(\mathcal{W}, \mathcal{U})$ from the original one does not exceed

$$\frac{\beta_{\mathcal{W}}^2}{2 \cdot 3^n} + \frac{3^l \beta_{\mathcal{U}}}{3^n - \beta_{\mathcal{U}}}.$$

This probability is negligible under the chosen l, n and reasonable restrictions on $\beta = \beta_{\mathcal{W}} + \beta_{\mathcal{U}}$.[18]

In parallel with the simulation of $(\mathcal{W}, \mathcal{U})$, S creates an environment for the IND-CCA attack. The simulator picks $b \xleftarrow{R} \{0, 1\}$ and passes to A the given ciphertext c^* as a result of encryption of m_b. The simulator virtually links m_b with c^* using its simulating potential.

More specifically, if $\mathcal{W}$ gets a query (s^*, m_b) and it turns out that the polynomials $r^* = s^*/h$ and $t^* = c^* - s^*$ are ternary (that is, small), then the simulator will assign $\mathcal{W}$ the answer t^* to the query (s^*, m_b). Such an assignment implies that `NTRU.Encr` with $r = r^*$ actually maps m_b to c^*.

By assigning $\mathcal{W}$: $(s^*, m_b) \mapsto t^*$, S violates the conditions of the simulation. But it does not matter: after receiving the query (s^*, m_b) the simulator stops interactions with A since the solution (r^*, t^*) of NTRU Problem has already found!

It does not matter what plaintext m is in (s^*, m). Only s^*, which reveals the solution (r^*, t^*), matters. If an appropriate s^* is encountered in arbitrary query (s^*, m), then the simulator also interrupts the simulation and returns the solution.

The simulator finds the solution (r^*, t^*) and stops the simulation in one more case: the oracle $\mathcal{U}$ receives a request (s^*, t^*) and it turns out that the polynomial $r^* = s^*/h$ is ternary. Interestingly, this case corresponds to the usual decryption c^* with a known f: $t^* \leftarrow c^*(1+3f)$, $s^* \leftarrow c^* - t^*$, $m_b \leftarrow \mathcal{U}(s^*, t^*)$.

Before recieving useful queries (s^*, m_i) or (s^*, t^*), S has to simulate $\mathcal{D}$ additionally to $(\mathcal{W}, \mathcal{U})$. This additional simulation is carried out as follows:

1. The simulator starts another list E into which inputs / outputs (m, c) of `NTRU.Encr` are placed. The list is initially empty.

[18] In NPQCC, it is supposed that $\beta < 2^{64}$.

2. For each query (s, m) to $\mathcal{W}$, the simulator calculates $r = s/h$. If r is ternary, then S calculates $c \leftarrow s + \mathcal{W}(s, m)$ and appends the pair (m, c) to E. The simulator actually predicts an encryption result $\texttt{NTRU.Encr}(h, m)$ which could get A. The prediction is possible because the input s reveals a random polynomial r used for encryption.

3. When processing the query c to $\mathcal{D}$, the simulator looks for a pair (m, c), which second element is prescribed, in E. If the pair is found, then S returns m, if not found — $\perp$. In other words, S decrypts only those ciphertexts that might have been recieved in a result of previous encryptions.

This simulation of $\mathcal{D}$ means that S virtually knows f, finds $t \leftarrow c(1 + 3f)$ and $s \leftarrow c - t$ and after that asks $\mathcal{U}$ with the query (s, t) assuming the answer $\perp$. This answer is correct if in the future the oracle $\mathcal{W}$ does not give the answer t to a query (s, m) with the prescribed s and some m. If $\beta_{\mathcal{D}}$ is the number of queries to $\mathcal{D}$, then the probability of the interesting event is not less than

$$\left(1 - \frac{1}{3^n}\right)^{\beta_{\mathcal{D}}-1} > 1 - \frac{\beta_{\mathcal{D}}}{3^n}.$$

In whole, the simulator conducts a simplified simulation of $(\mathcal{W}, \mathcal{U}, \mathcal{D})$, which can only be distinguished from the original one with a negligible probability

$$p = \frac{\beta_{\mathcal{W}}^2}{2 \cdot 3^n} + \frac{3^l \beta_{\mathcal{U}}}{3^n - \beta_{\mathcal{U}}} + \frac{\beta_{\mathcal{D}}}{3^n}$$

up to the appearance of useful queries (s^*, m) or (s^*, t^*).

Let $\mathcal{E}$ be an event that a useful query appears. The occurence of $\mathcal{E}$ is a necessary condition for $\mathbf{Adv}^{\text{IND-CCA}}(A)$ to be non-zero. Indeed,

$$\mathsf{P}\{\hat{b} = b\} = \mathsf{P}\{\hat{b} = b \mid \mathcal{E}\}\mathsf{P}\{\mathcal{E}\} + \mathsf{P}\{\hat{b} = b \mid \bar{\mathcal{E}}\}(1 - \mathsf{P}\{\mathcal{E}\}).$$

If $\mathcal{E}$ does not occur, then the adversary submits neither queries (s^*, m), which could contain information about mapping m_b to c^*, nor queries (s^*, t^*), which could contain information about decryption of c^*. Instead, the adversary gets random unrelated answers $\mathcal{W}(s, m)$ to non-informative queries (s, m) or some derivatives from these answers. Thus, $\mathsf{P}\{\hat{b} = b \mid \bar{\mathcal{E}}\} = 1/2$,

$$\mathsf{P}\{\hat{b} = b\} = \frac{1}{2} + \mathsf{P}\{\mathcal{E}\}\left(\mathsf{P}\{\hat{b} = b \mid \mathcal{E}\} - \frac{1}{2}\right)$$

and $\mathbf{Adv}^{\text{IND-CCA}}(A) \leqslant \frac{1}{2}\mathsf{P}\{\mathcal{E}\}$.

Finally, the simulator solves NTRU Problem with a probability of at least

$$2\mathbf{Adv}^{\text{IND-CCA}}(A) - p.$$

This probability is high if the target PKE system is not IND-CCA secure. Since NTRU Problem is hard, IND-CCA security must hold. □

The irony is that the IND-CCA guarantees are too strong for MAM2. Indeed, in MAM2 we use PKE algorithms only for encryption of volatile session keys. Since an adversary does not control session keys, she has difficulties even with CPA (Chosen Plaintext Attack). But we suppose that the proposed algorithms can be used in a wider context, and in this context the strongest security guarantees might be useful. It is also important that guarantees are achieved at low expenses.

5.7 Security

5.7.1 The meet-in-the-middle attack

The original idea of the meet-in-the-middle attack on NTRU belongs to Odlyzko and it was firstly described in [117]. The attack allows to recover a private key from the corresponding public key. Some improvements of this attack were proposed in [115] and [180].

The main idea of the attack is to split the search space of the possible private keys into two parts and to search these parts separately in order to find the full private key.

To identify search space for the NTRU let use the following equation from 5.6.2:

$$h = 3g/(1+3f) \bmod q.$$

To simplify the further reasoning we assume that $F = 1+3f$, $H = h/3$. So the above equation has form:

$$F \cdot H = g.$$

Suppose F is $F_1 + F_2$ and rewrite the equation above:

$$F_1 \cdot H = g - F_2 \cdot H.$$

We can see that $F_1 \cdot H$ differs from $-F_2 \cdot H$ on polynomial g with small coefficients. If all coefficients of g equal zero, then we need to find classic collision. However, in our case we need to find *almost-collision*. In the original meet-in-the-middle attack, g is considered as a polynomial with binary coefficients. In our case, $g_i \in \{-1, 0, 1\}$, that makes this attack more complicated.

To find almost-collisions, we need an auxiliary function $a(x)$. This is a sign function that returns vector of $\mathbf{I}\{x_i > 0\}$ where $i \in 1, \ldots, n$. $a(F_1 \cdot H), a(-F_2 \cdot H)$ are the *addresses* for F_1 and F_2 polynomials respectively. The first step of attack is to calculate all possible variants of $F_1 \cdot H$ and place each F_1 in `box` with address $a(F_1 \cdot H)$. In the second step, we calculate $-F_2 \cdot H$ and check if there are any elements in a *box* with address $a(-F_2 \cdot H)$. If it is true, then we check that all coefficients of $(F_1 + F_2) \cdot H$ are from $\{-1, 0, 1\}$. If it is true, then we have found either private key or its rotation. In the other case, since g is not zero polynomial, then we need to check boxes with all possible distinct addresses $a(-F_2 \cdot H + g)$ for F_2. If we still have not found private key, then we check the next F_2

Let us calculate the search space for this attack. In our case, the polynomials F and G do not have prescribed numbers of 0's and ± 1's, in contrast to common NTRU cryptosystems. So the search space for F_1 and F_2 can be defined as $O(3^n/2)$ polynomials. However, this estimation does not take into account equivalent keys. As key F is equivalent to all of the rotated keys $x^i \cdot F$ and to the negations $-x^i \cdot F$ so the final estimation of iterations for the finding collision is:

$$L = \frac{2 \cdot 3^{n/2}}{\sqrt{2n}}.$$

It should be noted that the cost of each iteration can differ for various attack implementation. For example, we can store not only F_1 in boxes, but $F_1 \cdot H$ too. It decreases time for key search but increases required space.

5.7.2 The lattice reduction attack

BKZ-2.0 is the best lattice reduction algorithm known in practice for high dimension lattice [64]. This algorithm is improved variant of the block Korkine-Zolotarev (BKZ) algorithm. In turn, BKZ is a blockwise generalization of LLL algorithm.

BKZ-2.0 requires solving the SVP problem in a smaller dimension and has parameters which one can set to obtain different run-times and output qualities. Such parameters include the block size β, which is upper-bounded by the rank of the lattice, and the number of rounds R. The run-time grows at least exponential in β.

In general BKZ-2.0 algorithm is as follows. First, the input lattice is LLL reduced, giving a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of a lattice L. After that, the algorithm executes the following round function R times. In each round we iterate the index i from one to $d - \beta$, and for each value of i we take the β-dimensional projected lattice generated by the basis vectors $\mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta-1}$ projected onto the orthogonal space spanned by the first $i - 1$ basis vectors. A small vector is obtained in the projection of this lattice, and the resulting vector is inserted into the main lattice basis at the i-th position. After insertion, we perform LLL on the truncated basis to remove linear dependencies.

Several algorithms can be used to realize the SVP oracle inside BKZ-2.0. In particular, the search for the small vector in the projected lattice is performed by the enumeration [151] or sieving algorithms [22]. There is not yet a consensus in the cryptographic community as to which algorithm to choose. The most commonly considered SVP oracle is sieving.

The fastest sieving algorithms run in time $2^{c\beta+o(\beta)}$, where $c = 0.292$ in the classic setting [22] and $c = 0.265$ in the post-quantum setting with Grover speedups [133]. Usually, the value $o(\beta)$ is omitted or replaced by a constant based on experimental evidence.

BKZ-2.0 proceeds by repeatedly calling an SVP oracle for computing the shortest vector on a projected lattice of dimension β. It is known that the number of calls to that oracle remains polynomial, yet it has been established that only the first few calls make significant progress [63]. Some cost models assume that one BKZ-2.0 call costs as much as β or $8d$ calls to the SVP oracle [7]. Alternatively, the popular cost model from [10] assumes the cost of BKZ-2.0 to be the same as one SVP oracle call. The last estimation is a pessimistic estimation (from the defender's point of view).

After performing BKZ-2.0 reduction with block size β the shortest vector $\mathbf{v}$ in the transformed lattice basis will have norm $\delta^d \det(\mathcal{L}(L))^{1/d}$, where δ is root Hermite factor. Increasing the parameter β leads to a smaller δ but also leads to an increase in run-time.

If the basis B is BKZ-2.0 reduced with block size β we can assume [63] the following relation between the block size and the root Hermite factor

$$\delta = \left(\frac{(\pi\beta)^{1/\beta}\beta}{2\pi e} \right)^{1/(2(\beta-1))}. \tag{5.1}$$

According to the experiments in [79] for the d-dimensional NTRU lattices the corresponding Hermite factor can be specified by

$$\frac{\sqrt{d/2\pi e} \cdot \det(\Lambda_q(B))^{1/d}}{\|\mathbf{v}\|} \approx \tau\delta^d \tag{5.2}$$

where $\tau = 0.4$ and $\mathbf{v}$ is the actual shortest vector.

Note, J. Hoffstein et al. [113] give an estimations of the root Hermite factor for proposed parameter sets of NPQCC/NTRUEncrypt with $d = 2n$ and $\|\mathbf{v}\| \approx \sqrt{4n/3}$. These estimations can be obtained for $\tau = 0.35$.

Let us estimate the norm of the shortest vector of the final scheme under the assumption that the polynomials f and g are chosen randomly and uniformly from the set of possible values (disregarding the requirement of reversibility of polynomials).

Determine the probability $p_r = \mathsf{P}\{\|\mathbf{v}\| < r\}$. This probability depends on the parameters r, n and can be defined as follows:

$$p_r = \mathsf{P}\left\{\|\mathbf{v}\|^2 < r^2\right\} = \mathsf{P}\left\{(1+3f_0)^2 + \sum_{i=1}^{n-1}(3f_i)^2 + \sum_{i=0}^{n-1} g_i^2 < r^2\right\} = \\ = \frac{1}{3}(p_{r,1} + p_{r,16} + p_{r,4}), \tag{5.3}$$

where

$$p_{r,k} = \mathsf{P}\left\{\sum_{i=1}^{n-1}(3f_i)^2 + \sum_{i=0}^{n-1} g_i^2 < r^2 - k\right\} = \\ = \sum_{\substack{i=0,1,\ldots,n-1 \\ j=0,1,\ldots,n \\ 9i+j<r^2-k}} \binom{n-1}{i}\left(\frac{2}{3}\right)^i\left(\frac{1}{3}\right)^{n-1-i}\binom{n}{j}\left(\frac{2}{3}\right)^j\left(\frac{1}{3}\right)^{n-j} = \\ = \frac{1}{3^{2n-1}} \sum_{\substack{i=0,1,\ldots,n-1 \\ j=0,1,\ldots,n \\ 9i+j<r^2-k}} \binom{n-1}{i}\binom{n}{j} 2^{i+j}. \tag{5.4}$$

For a given n, let r_n is such a value that $p_{r_n} < 2^{-256}$, i.e. $\|\mathbf{v}\| \geqslant r_n$ with a probability close to 1. Then from (5.2) and the relation $\det(\Lambda_q(B)) = q^n$ it follows that for the final scheme, in the worst case, the root Hermit factor takes the value:

$$\delta \approx \left(\frac{\sqrt{qn/\pi e}}{\tau r_n}\right)^{1/(2n)}. \tag{5.5}$$

To estimate the running time of BKZ-2.0 for selecting parameters q and n for the target NTRU scheme we use the following approach:

1. Use the relations (5.3), (5.4) in order to estimate r_n.
2. Use the relation (5.5) for $\tau = 0.35$ in order to determine the root Hermit factor δ.
3. Solve the relation (5.1) for β in order to establish the required block size β.
4. Estimate the running time $T_1 = 2^{0.292\beta}$ in the classic setting and $T_2 = 2^{0.265\beta}$ in the post-quantum setting.

Table 5.6 gives the estimations of the evalueted values for the final scheme for $n = 512, 768, 1024$ and $q = 12289$.

Table 5.6: The estimations of the evalueted values

n	512	768	1024
r_n	39.31	53.18	64.67
δ	1.00404	1.00263	1.00195
β	392	712	1060
T_1	$2^{114.5}$	$2^{207.9}$	$2^{309.5}$
T_2	$2^{103.9}$	$2^{188.9}$	$2^{280.9}$

5.7.3 The hybrid attacks

In 2007, N. Howgrave-Graham [115] developed the so-called hybrid attack on NTRUEncrypt scheme. The attack was proposed as a combination of a lattice reduction attack and a meet-in-the-middle combinatorial search attack. Several papers [113, 171] claim that the hybrid attack is the best-known attack on NPQCC/NTRUEncrypt. Although work [183] suggests that the power of a hybrid attack is over-estimated.

A generalized and latest version of the hybrid attack is presented in [183]. While previous versions of the hybrid attack suffer from using unnecessary and oversimplifying assumptions, which distort the accuracy of the security estimates, version [183] is based on reasonable assumptions.

The task of the hybrid attack is to find a (unique) shortest non-zero vector $\mathbf{v}$ in a q-ary lattice Λ, given a basis of Λ of the form

$$B' = \begin{pmatrix} B & C \\ 0 & I_r \end{pmatrix} \in \mathbb{Z}^{d \times d},$$

where $0 < r < d$ is the meet-in-the-middle dimension, $B \in \mathbb{Z}^{(d-r)\times(d-r)}$ and $C \in \mathbb{Z}^{(d-r)\times r}$. In [183] was shown that for q-ary lattices, where q is prime, one can always construct a basis of this form for a suitable r, provided that the determinant of the lattice is at most q^{d-r}.

The main idea of the attack is the following (see [183]). Let $\mathbf{v}$ be a shortest non-zero vector contained in the lattice Λ. The vector $\mathbf{v}$ is split into two contiguous parts $\mathbf{v} = (\mathbf{v}_l, \mathbf{v}_g)$ with $\mathbf{v}_l \in \mathbb{Z}^{d-r}$ and $\mathbf{v}_g \in \mathbb{Z}^{d-r}$. The second part $\mathbf{v}_g$ is recovered by using Odlyzko's meet-in-the-middle approach [117], while the first part $\mathbf{v}_l$ is recovered with lattice techniques by using the Babai's Nearest Plane algorithm [16].

The Nearest Plane algorithm is a BDD (see 5.2.2) algorithm. The input for the algorithm is a lattice basis $B \subset \mathbb{Z}^m$ of a full-rank lattice Λ and a target vector $\mathbf{t} \in \mathbb{R}^m$. The corresponding output of the algorithm is a vector $\mathbf{e} \in \mathbb{R}^m$ such that $\mathbf{t} - \mathbf{e} \in \Lambda(B)$. The output is denoted $\mathtt{NP}_B(t)$.

The idea of the meet-in-the-middle search is that instead of guessing $\mathbf{v}_g$ directly in a large set M of possible vectors, one guesses sparser vectors $\mathbf{v}'_g$ and $\mathbf{v}''_g$ in a smaller set N of vectors such that $\mathbf{v}'_g + \mathbf{v}''_g = \mathbf{v}_g$.

The parameters used in the attack include the lattice dimension d, the meet-in-the-middle dimension r, the block size β, the root Hermite factor δ corresponding to β, the lattice basis B', the partially reduced lattice basis B and other. The main attack parameters are r and β.

In general, the attack is as follows. Assume that $\mathtt{NP}_B(C\mathbf{v}_g) = \mathbf{v}_l$. First, we guess vectors $\mathbf{v}'_g$ and $\mathbf{v}''_g$ in the set N. We then compute $\mathbf{v}'_l = \mathtt{NP}_B(C\mathbf{v}'_g)$ and $\mathbf{v}''_l = \mathtt{NP}_B(C\mathbf{v}''_g)$. We hope that

if $\mathbf{v}'_g + \mathbf{v}''_g = \mathbf{v}_g$, then also $\mathbf{v}'_l + \mathbf{v}''_l = \mathbf{v}_l$. The probability that this additive property holds is one crucial element in the runtime analysis of the attack [183]. In order to detect when this property holds during the attack, we store $\mathbf{v}'_g$ and $\mathbf{v}''_g$ in (hash) boxes whose addresses depend on $\mathbf{v}'_l$ and $\mathbf{v}''_l$, respectively, such that they collide in at least one box. In order to increase the chance of recovering $\mathbf{v}_l$ being successful we perform a lattice reduction of B as precomputation. BKZ 2.0 (see 5.7.2) with the block size β is used for a lattice reduction.

The total runtime of the complete hybrid attack involves the runtime of lattice reduction, the runtime of the actual hybrid attack, and the success probability. All these quantities depend on the meet-in-the-middle dimension r and root Hermite factor δ corresponding to the applied block size β, i.e. the quality of the basis B [183].

Th. Wunderer [183] shows that under sufficient parameters the attack is successful. He provides under-estimates and over-estimates of the expected runtime in the success case, where the under-estimates account for possible improvements which have not yet shown to be applicable. In particular, the evaluated estimates of the runtime (security levels) for recent parameter sets of NPQCC/NTRUEncrypt are presented in Table 5.7. These estimates show that the security levels against the hybrid attack claimed in [113] are lower than the actual security levels for all parameter sets of NPQCC/NTRUEncrypt. In addition, the results of [183] shows that while for all of the analyzed parameter sets the hybrid attack outperforms a pure lattice reduction attack, it does not perform better than a purely meet-in-the-middle attack.

Table 5.7: Security levels in bits against the hybrid attack on NPQCC/NTRUEncrypt

n	401	439	593	743
Security levels of [113]	116	133	201	272
Security levels (under/over) of [183]	145/162	165/182	249/267	335/354

A number of works have attempted to analyze the complexity of the hybrid attack in the post-quantum setting. The main idea is to replace the meet-in-the-middle search with Grover search. However, it is currently unknown how to achieve a quantum speedup for collision search in a data locality-sensitive model [171].

Thus, it can be assumed that the hybrid attack on the final scheme does not perform better than a pure lattice reduction attack (see 5.7.2) and a pure meet-in-the-middle attack.

5.7.4 Other attacks

There are several type of attacks on the NTRU scheme that are not so good as the meet-in-the-middle, hybrid or BKZ attacks. Among them, we should highlight the following: combinatorial attack, subfield attack and algebraic attack.

The combinatorial attack uses a specific structure of polynomials f and g. Usually, this structure combines with the meet-in-the-middle attack. In our case these polynomials are F and g. Since they have not specific structure this attack becomes ordinary the meet-in-the-middle attack.

The subfield attacks were described in [2], [130] and related with so-called "overstretched" parameters. This means that this attack can be applied only on NTRU cryptosystems with $q > n^{2.83}$. So we can conclude that our cryptosystem is out of reach for such an attack.

The algebraic attacks on NTRU are not very effective for a now. Some of them [42] are applied only on cryptosystems where q is a power of two, some of them [75] are worse than, for example, MITM attack. So we can say that our cryptosystem is secure against these attacks.

5.8 IOTA addresses and public keys

Before encrypting using Alice's public key pk_A, Bob needs to get this key. When delivering pk_A, its integrity and authenticity must be ensured: Bob must have guarantees that 1) the key really belongs to Alice and 2) the key has not been changed during the delivery process. In other words, an *authenticated channel* must be used for delivering.

Usually, hierarchical PKIs (Public Key Infrastructures) are used for key distribution. In their simplest form, they work as follows. A third party, Trent, to which Alice and Bob trust, appears. Trent binds Alice's key pk_A with her identification data Id_A issuing a certificate $\texttt{Cert}_T(Id_A, pk_A)$. In the certificate, the information about Alice and her key is accompanied by Trent's signature. Bob verifies the signature using Trent's public key pk_T (we assume that it was delivered in advance in an authenticated manner) and extracts pk_A from the certificate. By distributing keys in the form of certificates, Trent creates implicit authenticated channels for delivering public keys to subordinate parties. Interestingly, MAM2 already provides authenticated channels since MAM2 messages are signed by senders.

In the case of MAM2, certificate management can be organized as follows:

1. Trent creates a MAM2 channel for distributing certificates. A channel's public key (`chid`) stands as an IOTA address.

2. Trent signs channel messages including a message that contains Alice's public key. This message may have the following form:

   ```
   message Cert {
     trytes name;
     tryte pubkey_ntru[3072];
   }
   ```

   The field `name` describes Alice's identification data, the field `pubkey` contains her public key. The correctness of the binding between `name` and `pubkey` is the responsibility of Trent. The string `name` can be empty if an anonymous key-centric PKE, such PKEs are very popular in cryptocurrencies, is intended;

3. Bob verifies the channel signatures. If the signature of `Cert` is correct, then Bob extracts the public key `pubkey` and encrypts the messages on it for the recipient `name`.

Of course, for the certificate distribution to be secure, Bob has to trust Trent or, more precisely, his channel. If Bob trusts Trent, then he trusts his messages and, therefore, public keys contained in these messages.

In principle, Trent can publish in his channel not only PKE but also DS keys. For example, Trent publishes `chid` of Alice's channel and then Bob expands his trust to Alice, to messages published in her channel, to other public keys, to other channels, etc.

The extended certificate might have the following format:

```
message Cert {
  trytes name;
  oneof pubkey {
    tryte chid[81] = 1;
    tryte ntru[3072] = 2;
}
```

We can abandon the hierarchy by allowing to publish Trent's keys in Alice's channel. Now each party can approve other parties. We obtain an analogue of *Web of Trust* in PGP style.

Alice herself can serve as Trent, that is, publishe her public key in her own channel. Both PKE and DS keys can be published. In the latter case, Alice issues a so-called *self-signed certificate*. This certificate proves that Alice really knows the corresponding private key, this proof can be useful in certain situations.

Let us imagine that you have developed DS algorithms that run using the same keys as the PKE (NTRU) algorithms. Let Alice declare a hash value of her DS-PKE public key as a `chid` of her channel. When Alice creates a channel she immediately publishes in it the public key. Thereby, a self-signed certificate is issued, with which Bob can both send confidential messages to Alice and verify her messages. In fact, a universal cryptographic point of interaction with Alice is created.

Conclusion

1. An overview of MAM/MAM2-related topics of quantum-proof cryptography and the IOTA infrastructure is conducted. The main expectations about future MAM2 features are presented.

2. An overview of basic approaches to the construction of authenticated encryption (AE) algorithms, sponge-based constructions and modes of their operation is conducted. The `Sponge`, `Duplex`, `MonkeyDuplex`, full-state keyed sponge (FKS), full-state keyed duplex (FKD) constructions and the `SpongeWrap`, `DuplexWrap`, `MonkeyWrap` modes are considered.

3. Generic and post-quantum security of basic sponge-based constructions are considered. It is shown that FKS and FKD are weak in the quantum model. Other basic sponge constructions, when used properly, are secure in both the classical and quantum models.

4. A comparison between MAM AE and general sponge-based AE algorithms is conducted. To protect messages of a fixed format, MAM AE employs the `Duplex` construction similar to `SpongeWrap` in the overwrite mode without padding. The `Duplex` construction and the `SpongeWrap` mode was publicly studied and was shown to be secure.

5. The applicability of universal attacks against MAM AE is investigated. Security of MAM AE against "time-memory tradeoff" and "forgery" attacks is shown. It is suggested that a MAM message header has to be padded up to a full block to improve security against "rollback" attacks.

6. An informal specification of sponge-based MAM2 algorithms is proposed. MAM2 algorithms are based on the `Duplex` construction similar to `SpongeWrap`. The overwrite mode with the classical padding and additional control (frame) trits is used. A post-quantum attacker has to perform roughly 2^{128} quantum operations to break the algorithms. The sponge capacity of MAM2 AE is almost doubled in comparison to MAM AE. As a consequence, MAM2 AE is almost twice faster than MAM AE.

7. An overview of modern hash-based cryptography is conducted. The main emphasis is on digital signature (DS) algorithms.

8. MAM DS algorithms are analyzed. They follow the Merkle signature scheme (MSS) with underlying Winternitz one-time signatures (WOTS). An instantiation of WOTS encapsulated in MAM DS uses the normalization approach instead of usual checksums.

9. In MAM DS, there are three security levels. It is detected that dimensions at some levels are unbalanced. The rough security estimates at each of the levels are (in the form

"level — Pre-Quantum strength /Post-Quantum strength"): I — $2^{64}/2^{42}$, II — $2^{128}/2^{85}$, III — $2^{192}/2^{128}$.

10. MAM2 DS — another instantiation of the MSS[WOTS] scheme — is designed. Dimensions of MAM DS are balanced. Although the normalization seems to be a good design idea with good perspectives from the theoretical point of view, we, nevertheless, decide to use checksums due to their simplicity. Post-quantum security of MAM2 DS is estimated. The Pre-Quantum / Post-Quantum strengths of MAM2 DS are roughly 2^{192} / 2^{128}.

11. The security of WOTS is proved. Unlike WOTS+ and other similar alternatives, the basic WOTS scheme isn't changed. Instead, non-standard computational problems regarding the involved hash functions are used.

12. The MAM2 expectations are converted into a set of rules that cover interactions between MAM2 entities and objects that support these interactions.

13. The concept of MAM2 is developed. According to this concept, messages are transmitted through endpoints which are parts of channels. Using the cascade (channel keys, endpoint keys) a channel owner can sign a fairly large number of messages and, therefore, overcome well-known inherent one-timeness of hash-based signatures. A sender can choose groups of recipients and deliver them the required key material through so-called keyloads. The PSK (PreShared Key) keyload is developed.

14. A special data definition language called ProtoBuf3 is developed. It is based on the well-known Google Protocol Buffers notation. ProtoBuf3 supports ternary streams. It is less universal than the basic prototype but somewhat more laconic. ProtoBuf3 contains cryptographic elements with the help of which one can quite easily describe the cryptographic processing of messages.

15. Formats of MAM2 messages are developed and presented using the Protobuf3 language. A message is divided into packets. Packets are protected independently. A packet can contain a checksum which is either a MAC or a signature.

16. The basic specification of MAM2 is developed. The specification is quite flexible and can support additional keyloads and other features in future.

17. An overview of the main Quantum-Proof cryptographic platforms is conducted. These platforms are LBC (Lattice-Based), CBC (Code-Based), MBC (Multivariate-Based) and other. The LBC platform is found the most suitable for building PKE (Public-Key Encryption) algorithms.

18. An overview of NTRU and LWE cryptography, the main branches of LBC, was conducted. The NTRU branch is found the most suitable.

19. NTRU-like PKE algorithms are developed. In the algorithms, NTT-friendly long-term parameters are used, encryption failures are suppressed. Core algorithms are accompanied with the previously developed AE (Authenticated Encryption) algorithms to achieve the IND-CCA security. We call our design (a bit funny) "*A NTT-friendly NTRU-like PKE with IND-CCA through AE and without decryption failures*".

20. The security of the developed PKE algorithms against the meet-in-the-middle attack, the lattice reduction attack, the hybrid attack, and some other standard attacks is estimated. It is shown that the algorithms are secure.

21. We discussed how to securely deliver PKE public keys in the IOTA infrastructure. We proposed to bind a public key with identification data about its owner in a special data structure, which is traditionally called a certificate, and publish certificates in channels of trusted entities. Certificates can cover not only PKE but also DS keys. Using DS certificates, we can create trust relations between different entities of MAM2.

22. The extended specification of MAM2 is developed. The extended specification additionally includes the designed PKE algorithms. Some existing Protobuf3 constructions are simplified are strengthened.

23. The `Spongos` layer for basic cryptoprocessing of strictly formatted data is developed. All top-level cryptographic algorithms are switched to work with `Spongos`.

24. The basic and extended MAM2 specifications are implemented in C language.

Bibliography

[1] Abed F., Forler Ch., Lucks S. General Classification of the Authenticated Encryption Schemes for the CAESAR Competition. Computer Science Review, Volume 22, 2016, pp. 13–26.

[2] Albrecht M., Bai S,,Ducas L. A subfield lattice attack on overstretched NTRU assumptions cryptanalysis of some FHE and graded encoding schemes, 2016.

[3] Aggarwal D., Joux A., Prakash A., Santha M. A new public-key cryptosystem via Mersenne numbers. Cryptology ePrint Archive, Report 2017/481, version:20170530.072202, 2017.

[4] Aggarwal D., Joux A., Prakash A., Santha M. A new public-key cryptosystem via Mersenne numbers. November 30, 2017

[5] Agrawal M., Chang D., Sanadhya S. A New Authenticated Encryption Technique for Handling Long Ciphertexts in Memory Constrained Devices. Cryptology ePrint Archive, Report 2015/331, 2015. Avail. at: `https://eprint.iacr.org/2015/331.pdf`.

[6] Akiyama K., Goto Y. A Public-key Cryptosystem using Algebraic Surfaces. In: Proc. of PQCrypto'06, pp. 119–138, (2006), Avail. at `http://postquantum.cr.yp.to/`.

[7] Albrecht M. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. EUROCRYPT 2017, Part II, LNCS 10211, Springer, Heidelberg, 2017, pp. 103–129.

[8] Albrecht M., Faugere J.-C., Farshim P., Herold G., Perret L. Polly cracker, revisited. Cryptology ePrint Archive, Report 2011/289, 2011. Avail at.: `https://eprint.iacr.org/2011/289`.

[9] Albrecht M., Cid K., Paterson K., Tjhai C., Tomlinson M. NTS-KEM: A code-based key-encapsulation mechanism submitted to NIST Post-Quantum Cryptography. NPQCC submission. Avail. at: `https://nts-kem.io/`.

[10] Alkim E., Ducas L., Poppelmann Ph., Schwabe P. Post-quantum key exchange — A new hope. In: 25-th USENIX Security Symposium, USENIX Security 16, USENIX Association, 2016, pp. 327–343.

[11] Andreeva E., Daemen J., Mennink B., Assche G. Security of keyed sponge constructions using a modular proof approach. FSE 2015, LNCS 9054, Springer, 2015, pp. 364–384.

[12] Aragon N., Blazy O., Deneuville J.-C., Gaborit P., Hauteville A., Ruatta O., Tillich J.-P., Zémor G. LAKE — Low rAnk parity check codes Key Exchange. NPQCC submission. Avail. at https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/LAKE.zip.

[13] Aumasson J.P., Endignoux G., Clarifying the subset-resilience problem. Cryptology ePrint Archive, Report 2017/909, 2017. Avail. at: https://eprint.iacr.org/2017/909.pdf.

[14] Aumasson J. P., Endignoux G. Gravity-SPHINCS. NIST competition submission. Avail. at: https://github.com/gravity-postquantum/gravity-sphincs/blob/master/Supporting_Documentation/submission.pdf.

[15] Galbraith S., Dwarakanath N. Efficient sampling from discrete Gaussians for lattice-based cryptography on a constrained device. Avail. at: https://www.math.auckland.ac.nz/~sgal018/gen-gaussians.pdf, 2018.

[16] Babai L. On Lovasz' lattice reduction and the nearest lattice point problem. Combinatorica, 6(1), 1986, pp. 1–13.

[17] Babbage S. Improved exhaustive search attacks on stream ciphers. European Convention on Security and Detection, IEE Conference publication No. 408, IEE, 1995, pp. 161–166.

[18] Banks W., Shparlinski I. A Variant of NTRU with Non-invertible Polynomials. In: Progress in Cryptology – INDOCRYPT'2002. INDOCRYPT 2002. LNCS 2551. Springer, 2002.

[19] Banerjee A., Peikert C., Rosen A. Pseudorandom Functions and Lattices. Advances in Cryptology – EUROCRYPT'2012. EUROCRYPT 2012. LNCS 7237. Springer, 2012, pp. 719–737.

[20] Banerjee T., Hasan M. Energy consumption of candidate algorithms for NIST PQC standards. Technical report, Centre for Applied Cryptographic Research (CACR) at the University of Waterloo, 2018. Avail. at: http://cacr.uwaterloo.ca/techreports/2018/cacr2018-06.pdf.

[21] Barg S. Some New NP-Complete Coding Problems, 1994. Avail. at: https://www.researchgate.net/publication/266919095_Some_new_NP-complete_coding_problems.

[22] Becker A., Ducas L., Gama N., Laarhoven Th. New directions in nearest neighbor searching with applications to lattice sieving. In: 27-th SODA, ACM-SIAM, 2016, pp. 10–24.

[23] Bellare M., Namprempre Ch. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. ASIACRYPT 2000, LNCS 1976, 2000, pp. 531–545.

[24] Bellare M., Rogaway P. Collision-Resistant Hashing: Towards Making UOWHFs Practical. Advances in Cryptology — CRYPTO '97. CRYPTO 1997. LNCS 1294. Springer, Berlin, Heidelberg. Avail. at: http://web.cs.ucdavis.edu/~rogaway/papers/tcr-hash.pdf.

[25] Berlekamp E., McEliece R., van Tilborg H. On the inherent intractability of certain coding problems, IEEE Transactions on Information Theory, 24(3), 1978, pp. 384–386. Avail. at: https://authors.library.caltech.edu/5607/1/BERieeetit78.pdf.

[26] Bernstein D., Buchmann J., Dahmen E. Post-Quantum Cryptography. Springer, 2009.

[27] Bernstein D., Chou T., Lange T., von Maurich I., Misoczki R., Niederhagen R., Persichetti E., Peters C., Schwabe P., Sendrier N., Szefer J., Wang W. Classic McEliece: conservative code-based cryptography NPQCC submission. Avail. at https://classic.mceliece.org/nist/mceliece-20171129.pdf.

[28] Bernstein D. et al. SPHINCS: Practical Stateless Hash-Based Signatures. In: Oswald E., Fischlin M. (eds) Advances in Cryptology – EUROCRYPT 2015. EUROCRYPT 2015. LNCS 9056. Springer, Berlin, Heidelberg.

[29] Bernstein D. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?, 2016. Avail. at: https://cr.yp.to/hash/collisioncost-20090517.pdf.

[30] Bertoni G., Daemen J., Peeters M., Van Assche G. Cryptographic sponge functions. Avail. at: https://keccak.team/files/CSF-0.1.pdf, 2011.

[31] Bertoni G., Daemen J., Peeters M., Van Assche G. Duplexing the sponge: single-pass authenticated encryption and other applications. In: Proceedings of the 18th international conference on Selected Areas in Cryptography, Toronto, ON, Canada (August 11 – 12), 2011, pp. 320–337.

[32] Bertoni G., Daemen J., Peeters M., Van Assche G. On the Indifferentiability of the Sponge Construction. In Nigel P. Smart, editor, EUROCRYPT, volume 4965 of Lecture Notes in Computer Science, pages 181 – 197. Springer, 2008.

[33] Bertoni G., Daemen J., Peeters M., Van Assche G. On the Security of the Keyed Sponge Construction. Symmetric Key Encryption Workshop, 2011.

[34] Bertoni G., Daemen J., Peeters M., Van Assche G. Permutation-Based Encryption, Authentication and Authenticated Encryption. Directions in Authenticated Ciphers, 2012.

[35] Bertoni G., Daemen J., Peeters M., Van Assche G., Van Keer R. CAESAR submission: Keyak v1. Avail. at: https://keccak.team/obsolete/Keyak-1.2.pdf, 2014.

[36] Bertoni G., Daemen J., Peeters M., Van Assche G., Van Keer R. CAESAR submission: Keyak v2. Avail. at: https://competitions.cr.yp.to/round3/keyakv22.pdf, 2014.

[37] Bertoni G., Daemen J., Peeters M., Van Assche G., Van Keer R. CAESAR submission: Ketje v2. Avail. at https://competitions.cr.yp.to/round3/ketjev2.pdf, 2016.

[38] Beullens W., Luengo I., Alperin-Sheriff J. Official comments — DME. Avail. at: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/DME-official-comment.pdf.

[39] Biryukov A., Shamir A. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers // Advances in Cryptology, proceedings of ASIACRYPT 2000, LNCS 1976, Springer-Verlag, 2000, pp. 1–13.

[40] Bleichenbacher D., Maurer U. Directed acyclic graphs, one-way functions and digital signatures. CRYPTO'94 Proceedings, 1994, pp. 75–82. Avail. at: https://link.springer.com/content/pdf/10.1007%2F3-540-48658-5_9.pdf.

[41] Bleichenbacher D., Maurer U. Optimal tree-based one-time digital signature schemes. In: Puech C., Reischuk R. (eds) STACS 96. STACS 1996. LNCS 1046. Springer, Berlin, Heidelberg. Avail. at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.2459&rep=rep1&type=pdf.

[42] Bourgeois G., Faugere J.-C. Algebraic attack on NTRU using Witt vectors and Groebner bases, Journal of Mathematical Cryptology, 3(3), 2009, pp. 205–214.

[43] Borowski M. The sponge construction as a source of secure cryptographic primitives. Military Communication Conference, France, 2013.

[44] Bos J.N., Chaum D. Provably unforgeable signatures. In: Advances in Cryptology — CRYPTO' 92. CRYPTO 1992. LNCS 740. Springer, Berlin, Heidelberg, pp. 1–14. Avail. at: https://link.springer.com/content/pdf/10.1007%2F3-540-48071-4_1.pdf.

[45] Boyer M., Brassard G., Høyer P., Tapp A. Tight bounds on quantum searching. Fortschritte der Physik, 46, 1998, pp. 493-–506. Avail. at: arXiv:quant-ph/9605034.

[46] Braeken A., Wolf Ch., Preneel B. A study of the security of Unbalanced Oil and Vinegar signature schemes. In: The Cryptographer's Track at RSA Conference 2005, LNCS 3376, Springer, 2005, pp. 29–43.

[47] Brassard G., Hoyer P., Tapp A. Quantum Algorithm for the Collision Problem. Third Latin American Symp. on Theoretical Informatics (LATIN'98), LNCS 1380, 1998, pp. 163–169.

[48] Brogan J., Baskaran I., Ramachandran N. Authenticating Health Activity Data Using Distributed Ledger Technologies. Version: 2018.04.25. Private communication.

[49] MacWilliams F., Sloane N. The Theory of Error-Correcting Codes. North-Holland Publishing Company, 1977.

[50] Bruinderink L., Hülsing A. "Oops, I Did It Again" — Security of One-Time Signatures Under Two-Message Attacks. In: Selected Areas in Cryptography — SAC 2017. SAC 2017. LNCS 10719. Springer, Cham, pp. 299–322. Avail. at: https://eprint.iacr.org/2016/1042.

[51] Buchmann J., Dahmen E., Hulsing A. XMSS — A Practical Forward Secure Signature Scheme based on Minimal Security Assumptions. Post-Quantum Cryptography. PQCrypto 2011. LNCS 7072. Springer, Berlin, Heidelberg, 2011, 2011, pp. 402–417. Avail. at: https://eprint.iacr.org/2011/484.pdf.

[52] Buchmann J., Cabarcas D., Göpfert D., Hülsing A., Weiden P. Discrete Ziggurat: A Time-Memory Trade-off for Sampling from a Gaussian Distribution over the Integers. In: Selected Areas in Cryptography – SAC 2013. SAC 2013. LNCS 8282. Springer, Berlin, Heidelberg, pp. 402–417. Avail. at: https://eprint.iacr.org/2013/510.pdf.

[53] Buchmann J. Dahmen E., Ereth S. Hülsing A., Rückert M. On the security of the Winternitz one-time signature scheme. International Journal of Applied Cryptography, 3, 2013, pp. 84–96.

[54] Buchmann J., Dahmen E., Klintsevich E., Okeya K., Vuillaume C. Merkle Signatures with Virtually Unlimited Signature Capacity. In: Applied Cryptography and Network Security. ACNS 2007. LNCS 4521. Springer, Berlin, Heidelberg, pp. 31–45. Avail. at: https://pdfs.semanticscholar.org/ce21/df2f09a724b931ec2a175d3fad841698b4e4.pdf.

[55] Buchmann J., Garcıa L., Dahmen E., Doring M., Klintsevich E. CMSS — An Improved Merkle Signature Scheme. Cryptology ePrint Archive, Report 2006/320, 2006. Avail. at: https://eprint.iacr.org/2006/320.pdf.

[56] Bernstein D. et al, Classic McEliece: conservative code-based cryptography, NIST submission, 2017. Avail. at: https://classic.mceliece.org/nist/mceliece-20171129.pdf.

[57] Courtois, N., Finiasz, M., Sendrier, N. How to Achieve a McEliece-based Digital Signature Scheme, 2001. Avail. at: https://www.iacr.org/archive/asiacrypt2001/22480158.pdf.

[58] Canetti R., Krawczyk H. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In: Advances in Cryptology — EUROCRYPT 2001. EUROCRYPT 2001. LNCS 2045. Springer, Berlin, Heidelberg, pp. 453–474.

[59] Chakraborty O., Faugere J.-C., Perret L. CFPKM: A Key Encapsulation Mechanism based on Solving System of non-linear multivariate Polynomials. NIST competition submission. Avail. at: https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[60] Chunsheng G. Integer version of ring-LWE and its applications. Cryptology ePrint Archive, Report 2017/641, 2017. http://eprint.iacr.org/2017/641.

[61] Chalkias K., Brown J., Hearn M., Lillehagen T., Nitto I., Schroeter T. Blockchained Post-Quantum Signatures. Cryptology ePrint Archive, Report 2018/658, 2018. Avail. at: https://eprint.iacr.org/2018/658.

[62] Chang D., Dworkin M., Hong S., Kelsey J., Nandi M. A Keyed Sponge Construction with Pseudorandomness in the Standard Model. The Third SHA-3 Candidate Conference, March 22–23, 2012.

[63] Chen Y. Reduction de reseau et securite concrete du chiffrement completement homomorphe. PhD thesis, 2013.

[64] Chen Y., Nguyen Ph. BKZ 2.0: Better lattice security estimates. ASIACRYPT 2011, LNCS 7073, Springer, 2011, pp. 1–20. Avail. at: http://www.iacr.org/archive/asiacrypt2011/70730001/70730001.pdf.

[65] Bardet N., Barelli E., Blazy O., Canto Torres R., Couvreur A., Gaborit P., Otmani A., Sendrier N., Tillich J.-P. BIG QUAKE BInary Goppa QUAsi–cyclic Key Encapsulation. NPQCC submission. Avail. at https://bigquake.inria.fr/files/2018/04/corrected_proposal.pdf.

[66] Cramer R., Ducas L., Peikert C., Regev O. Recovering Short Generators of Principal Ideals in Cyclotomic Rings. In: Advances in Cryptology — EUROCRYPT 2016. EUROCRYPT 2016. LNCS 9666. Springer, Berlin, Heidelberg, 2016.

[67] Czajkowski J., Bruinderink L., Hulsing A., Schaffner Ch., Unruh D. Post-quantum Security of the Sponge Construction. Cryptology ePrint Archive, Report 2017/771, 2017. Avail. at: https://eprint.iacr.org/2017/771.pdf.

[68] Czajkowski J., Bruinderink L., Hulsing A., Schaffner Ch., Unruh D. Post-quantum Security of the Sponge Construction. PQCrypto 2018: Post-Quantum Cryptography, pp. 185–204, 2018.

[69] Dahmen E., Okeya K., Takagi T., Vuillaume V. Digital Signatures out of Second-Preimage Resistant Hash Functions. In: Buchmann J., Ding J. (eds) Post-Quantum Cryptography. PQCrypto 2008. Lecture Notes in Computer Science, vol 5299. Springer, Berlin, Heidelberg. Avail. at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.559.9731&rep=rep1&type=pdf.

[70] del Pino R., Lyubashevsky V., Pointcheval D. The Whole is Less Than the Sum of Its Parts: Constructing More Efficient Lattice-Based AKEs. In: Security and Cryptography for Networks. SCN 2016. LNCS 9841. Springer, 2016.

[71] Denef J. The Diophantine Problem for Polynomial Rings of Positive Characteristic In: Proc. of Logic Colloquium'78, Stud. Logic Foundations Math., 97, pp. 131–145, North Holland, Amsterdam-New York, 1979

[72] Diffie W., Hellman M.E., New directions in cryptography. IEEE Transactions on Information Theory, 22:6, 1976, pp. 644–654. Avail. at: https://ee.stanford.edu/~hellman/publications/24.pdf.

[73] Ding J., Hodges T. Inverting HFE systems is quasi-polynomial for all fields. In: Advances in Cryptology – CRYPTO 2011. CRYPTO 2011, LNCS 6841, Springer, Heidelberg, 2011, pp. 724–742.

[74] Ding J., Petzoldt A., Wang L.C. The cubic simple matrix encryption scheme. In: PQC'14, LNCS 8772, 2014, pp. 76–87.

[75] Ding J., Schmidt D. Algebraic attack on lattice based cryptosystems via solving equations over real numbers, 2012.

[76] Ding J., Xie X., Lin X. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688, 2014. Avail. at: http://eprint.iacr.org/2012/688.

[77] Ding J., Yang B.-Y. Multivariate Public Key Cryptography. In: Bernstein D., Buchmann J., Dahmen E. (eds.) Post-Quantum Cryptography, 2009, Sprinher, pp. 193–241.

[78] Douglas P. How bundles are processed in the Tangle. Personal communication via `slack.com#collab-bsu-mam-x`, 2018.11.08.

[79] Ducas L., Durmus D., Lepoint T., Lyubashevsky V. Lattice signatures and bimodal gaussians. Crypto 2013, LNCS 8042, Springer, 2013.

[80] Ducas L., Nguyen P.Q. Faster Gaussian Lattice Sampling using Lazy Floating-Point Arithmetic. Avail. at: `https://homepages.cwi.nl/~ducas/LazyGaussSampling/DucasNguyen_Sampling.pdf`.

[81] Dunkelman O., Keller N. Treatment of the initial value in Time-Memory-Data Tradeoff attacks on stream ciphers. Information Processing Letters, 107(5), 2008, pp. 133–137.

[82] Dworkin M. Special Publication 800-38C: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. National Institute of Standards and Technology, U.S. Department of Commerce, May 2005.

[83] Elmaliah A., Gal A. IOTA signature scheme — Status Quo and improvements. Version: 2018.05.11. Private communication.

[84] Endignoux G. Design and implementation of a post-quantum hash-based cryptographic signature scheme. Master's Thesis, 2017. Avail. at: `https://gendignoux.com/assets/pdf/2017-07-master-thesis-endignoux-report.pdf`.

[85] Even Sh., Mansour Y. A construction of a cipher from a single pseudorandom permutation. Journal of Cryptology, Volume 10, Issue 3, June 1997, pp. 151–161.

[86] Faugere J.-C., Joux A. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Grobner bases. In: Advances in Cryptology – CRYPTO 2003, CRYPTO 2003, LNCS 2729, Springer, Heidelberg, 2003, pp. 44–60.

[87] Follath J. Gaussian sampling in lattice-based cryptography. Tatra Mountains Mathematical Publications, 60(1), pp. 1–23. Avail. at: `https://www.sav.sk/journals/uploads/0212094402follat.pdf`.

[88] Faugere J., Gauthier-Umana V., Otmani A., Perret L. A Distinguisher for High Rate McEliece Cryptosystems, IEEE Transactions on Information Theory, 2010(10), p. 331. Avail. at: `https://eprint.iacr.org/2010/331.pdf`

[89] Flajolet P., Odlyzko A. M. Random Mapping Statistics. In: Advances in Cryptology – EUROCRYPT '89. EUROCRYPT 89. LNCS 434, 1989, Springer, pp. 329–354.

[90] Fujisaki E., Okamoto T. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In: Advances in Cryptology – CRYPTO'99. CRYPTO 99. LNCS 1666, Springer, 1999, pp. 537-–554.

[91] Galbraith S., Petit C., Shani B., Yan B. Ti. On the security of supersingular isogeny cryptosystems. In: Advances in Cryptology –– ASIACRYPT'2016. ASIACRYPT 2016. LNCS 10031, 2016, Springer, pp. 63–91.

[92] Gama N., Nguyen P. Predicting lattice reduction. In: Advances in Cryptology — EUROCRYPT'2008. EUROCRYPT 2008. LNCS 4965. Springer, Berlin, Heidelberg, 2008, pp. 31–51.

[93] Garey M., Johnson D. Computers and intractibility: A guide to the theory of NP-completeness. New York: W.H. Freeman, 1979.

[94] Geiselmann W., Steinwandt R., Beth Th. Attacking the Affine Parts of SFLASH. IMA International Conference on Cryptography and Coding, 2001, pp. 355–359.

[95] Gentry C. Key recovery and message attacks on NTRU-composite. In: Advances in Cryptology – EUROCRYPT'01, EUROCRYPT 01. LNCS 2045. Springer, Berlin, Heidelberg, 2001, pp. 182–194.

[96] Gentry C., Peikert C., Vaikuntanathan V. How to Use a Short Basis: Trapdoors for Hard Lattices and New Cryptographic Constructions, 2007. Avail. at: `https://eprint.iacr.org/2007/432.pdf`

[97] Gligor V., Donescu P. Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes. In: Fast Software Encryption. FSE 2001. LNCS 2355. Springer, Berlin, Heidelberg, 2001, pp. 92–108.

[98] Gnedenko B. V. On the local limit theorem in the theory of probability. Uspekhi Mat. Nauk, 3:187–194, 1948.

[99] Goldreich O., Goldwasser S., Halevi S. Public-Key Cryptosystems from Lattice Reduction Problems. Advances in Cryptology – CRYPTO '97. CRYPTO 1997. LNCS 1294. Springer, Berlin, Heidelberg, 1997, pp. 112–131. Avail. at: `https://rd.springer.com/content/pdf/10.1007%2FBFb0052231.pdf`

[100] Goldreich O. Computational Complexity: A Conceptual Perspective. Cambridge University Press, 2008.

[101] Golich J. Cryptanalysis of alleged A5 stream cipher. In: Advances in Cryptology –– EUROCRYPT'97. EUROCRYPT 1997. LNCS 1233. Springer, Berlin, Heidelberg, pp. 239–255.

[102] Goldreich O., Micciancio D., Safra S., Seifert J. P. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. Information Processing Letters, 71:55–61, 1999.

[103] Gorodilova A., Agievich S., Carlet C., Gorkunov E., Idrisova V., Kolomeec N., Kutsenko A., Nikova S., Oblaukhov A., Picek S., Preneel B., Rijmen V., Tokareva N. Problems and solutions of the Fourth International Students' Olympiad in Cryptography NSUCRYPTO, 2018. Avail. at: `https://arxiv.org/abs/1806.02059`.

[104] Nguyen P.Q., Regev O. Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures. J. Cryptology, 2009, 22:139, pp. 150–169.. Avail. at: https://rd.springer.com/content/pdf/10.1007%2F11761679_17.pdf.

[105] Gaborit Ph., Murat G., Ruatta O., Zemor G. Low Rank Parity Check codes and their application to cryptography. The International Workshop on Coding and Cryptography (WCC 13), 2013. Avail. at http://www.selmer.uib.no/WCC2013/pdfs/Gaborit.pdf.

[106] Grover L. A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on the theory of computing, held in Philadelphia, PA, May 22-24, 1996, 212–219.

[107] Hamburg M. The Strobe protocol framework. Cryptology ePrint Archive, Report 2017/003, 2017. Avail. at: https://eprint.iacr.org/2017/003.pdf.

[108] Hashimoto Y. Multivariate Public Key Cryptosystems. In: Mathematical Modelling for Next-Generation Cryptography, Mathematics for Industry, vol. 29, 2018, Springer, pp. 17–43.

[109] Hellman M. A Cryptanalytic Time-Memory Tradeoff. IEEE Transactions on Information Theory, 26 (4), 1980, pp. 401–406.

[110] Hoffstein J., Pipher J., Silverman J. An Introduction to Mathematical Cryptography, 2008, Springer-Verlag, New York.

[111] Hoffstein J., Pipher J., Schanck J., Silverman J., Whyte W. Transcript secure signatures based on modular lattices. Avail. at https://eprint.iacr.org/2014/457.pdf.

[112] Hoffstein J., Pipher J., Whyte W., Zhang Z. A signature scheme from Learning with Truncation. Avail. at https://eprint.iacr.org/2017/995.pdf.

[113] Hoffstein J., Pipher J., Schanck J., Silverman J., Whyte W., Zhang Zh. Choosing Parameters for NTRUEncrypt. CT-RSA 2017: Topics in Cryptology — CT-RSA 2017, LNCS 10159, pp. 3 – 18.

[114] Hong J. Sarkar P. New Applications of Time Memory Data Tradeoffs. In: Advances in Cryptology — ASIACRYPT'2005. ASIACRYPT 2005. LNCS 3788. Springer, Berlin, Heidelberg, 2005, pp. 353–372.

[115] Howgrave-Graham N. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In: Advances in Cryptology — CRYPTO'2007. CRYPTO 2007. LNCS 4622. Springer, Berlin, Heidelberg, 2007, pp. 150–169.

[116] Howgrave-Graham N., Silverman J., Singer A., Whyte W. NAEP: Provable Security in the Presence of Decryption Failures. Cryptology ePrint Archive, Report 2003/172, 2003. Avail. at http://eprint.iacr.org/2003/172.

[117] Howgrave-Graham N., Silverman J., Whyte W. A meet-in-the-middle attack on an NTRU private key. Avail. at https://www.securityinnovation.com/uploads/Crypto/NTRUTech004v2.pdf.

[118] Hülsing A. SPHINCS+ — The smaller SPHINCS. Blog post, 2017. Avail. at: https://huelsing.wordpress.com/2017/12/04/sphincs+-the-smaller-sphincs/.

[119] Hülsing A. W-OTS+ — shorter signatures for hash based signature schemes. In: Progress in Cryptology — AFRICACRYPT 2013, LNCS 7918, Springer Berlin Heidelberg, 2013, pp. 173—188. Avail. at: http://huelsing.files.wordpress.com/2013/05/wotsspr.pdf.

[120] Hülsing A., Rijneveld J., Song F. Mitigating Multi-Target Attacks in Hash-based Signatures. Cryptology ePrint Archive, 2015/1256. Avail. at: https://eprint.iacr.org/2015/1256.pdf.

[121] Hülsing A., Rijneveld J., Schanck J., Schwabe P. High-Speed Key Encapsulation from NTRU. In: Cryptographic Hardware and Embedded Systems – CHES 2017. CHES 2017. LNCS 10529. Springer, 2017.

[122] Jao D., De Feo L. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Post-Quantum Cryptography. PQCrypto 2011. LNCS 7071. Springer, Berlin, Heidelberg, 2011, pp. 19–34.

[123] Jovanovic P., Luykx A., Mennink B. Beyond $2^{c/2}$ Security in Sponge-Based Authenticated Encryption Modes. Cryptology ePrint Archive, Report 2014/373, 2014. Avail. at: https://eprint.iacr.org/2014/373.pdf

[124] Jochemsz E. Goppa Codes and the McEliece Cryptosystem. Avail. at: https://klevas.mif.vu.lt/~skersys/vsd/crypto_on_codes/goppamceliece.pdf

[125] Jutla Ch. Encryption Modes with Almost Free Message Integrity. In: Advances in Cryptology — EUROCRYPT 2001. EUROCRYPT 2001. LNCS 2045. Springer, Berlin, Heidelberg, 2001, pp. 529–544.

[126] Kelly M., Kaminskyy A., Kurdzielz M., Lukowiak M., Radziszowski S. Customizable sponge-based authenticated Encryption using 16-bit S-boxes. Military Communications Conference, MILCOM 2015, 2015.

[127] Kipnis A., Patarin J., Goubin L. Unbalanced Oil and Vinegar Signature Schemes. In: Advances in Cryptology –— EUROCRYPT'99. EUROCRYPT 1999. LNCS 1592. Springer, Berlin, Heidelberg, 1999, pp. 206–222.

[128] Kipnis A., Patarin J., Goubin L. Unbalanced Oil and Vinegar Signature Schemes. Extended version, 2003. Avail. at: http://www.goubin.fr/papers/OILLONG.PDF.

[129] Kirchner P., Fouque P.-A. Comparison between subfield and straightforward attacks on NTRU. IACR Cryptology ePrint Archive report 2016/717, 2016.

[130] Kirchner P., Fouque P.-A. Revisiting lattice attacks on overstretched NTRU parameters. In: Advances in Cryptology — EUROCRYPT'2017. EUROCRYPT 2017. LNCS 10210. Springer, Cham, 2017, pp. 3–26.

[131] Krawczyk H. The Order of Encryption and Authentication for Protecting Communications (Or: How Secure is SSL?). In: Advances in Cryptology — CRYPTO 2001. CRYPTO 2001. LNCS 2139. Springer, Berlin, Heidelberg, 2001, pp. 310–331.

[132] Кушнеров А. Троичная цифровая техника. Ретроспектива и современность. Беэр-Шева, Израиль: ун-т им. Бен-Гуриона, 2005. 14 с. (Kushnerov A. Ternary digital devices. Retrospective and modernity. Israel, Beersheba, Ben Gurion University). Avail. at: http://314159.ru/kushnerov/kushnerov1.pdf.

[133] Laarhoven Th. Search problems in cryptography: From fingerprinting to lattice sieving. PhD thesis, Eindhoven University of Technology, 2015.

[134] Lafrance P., Menezes A. On the security of the WOTS-PRF signature scheme. Cryptology ePrint Archive, Report 2017/938, 2017. Avail. at: https://eprint.iacr.org/2017/938.

[135] Lagarias J., Odlyzko A. Solving low-density subset sum problems. Journal of the Association for Computing Machinery, 32(1), 1985, pp. 229–246.

[136] Lee W., Kim Y., No J., Post quantum signature scheme based on modified Reed-Muller code pqsigRM. NIST submission. Avail. at: http://ccl.snu.ac.kr/papers/pqsigrm/doc.pdf.

[137] Legernæs M. W. On the Development and Standardisation of Post-Quantum Cryptography — A Synopsis of the NIST Post-Quantum Cryptography Standardisation Process, its Incentives, and Submissions. Norwegian University of Science and Technology, 2018. Avail at: https://brage.bibsys.no/xmlui/handle/11250/2562554.

[138] Lenstra A., Lenstra H. Jr., Lovász L. Factoring polynomials with rational coefficients. Math. Ann., 261(4): 515-534, 1982.

[139] Levin L. The Tale of One-way Functions. Problems of Information Transmission (= Problemy Peredachi Informatsii), 39(1):92-103, 2003.

[140] Lidl R., Niederraiter H. Finite Fields, Cambridge University Press, 1997.

[141] Lindner R., Peikert C. Better key sizes (and attacks) for LWE-based encryption. In: CT-RSA, 2011, pp. 319–339.

[142] Liu F., Liu F. Universal Forgery and Key Recovery Attacks: Application to FKS, FKD and Keyak // Cryptology ePrint Archive, Report 2017/691, 2017. Avail. at: https://eprint.iacr.org/2017/691.pdf.

[143] Luengo I., Avendano M., Marko M. DME: a public key, signature and kem system based on double exponentiation with matrix exponents. NIST competition submission. Avail. at: https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[144] Lyubashevsky V., Peikert C., Regev O. On ideal lattices and learning with errors over rings. Journal of the ACM, 60(6):43, 2013, pp. 1–35.

[145] McGrew D., Viega J. The Galois/Counter Mode of Operation (GCM). Submission to NIST. Avail. at: http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf, 2004.

[146] Measurements of hash functions, indexed by machine eBACS: ENCRYPT Benchmarking of Cryptographic Systems. Avail.at: http://bench.cr.yp.to/results-hash.html. Last access: 16 July 2018.

[147] Menezes, A. J., van Oorschot P., Vanstone S. Handbook of Applied Cryptography. CRC Press. 2001. Sample chapters avail. at: http://cacr.uwaterloo.ca/hac/.

[148] Mennink B., Reyhanitabar R., Vizar D. Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption. In: Advances in Cryptology — ASIACRYPT 2015. ASIACRYPT 2015. LNCS 9453. Springer, Berlin, Heidelberg, 2015, pp. 465–490.

[149] Merkle R. Secrecy, Authentication and Public Key Systems. A certified digital signature, Ph.D. dissertation, Dept. of Electrical Engineering, Stanford University, 1979.

[150] Micciancio D., Regev O. Lattice-based Cryptography. In: Bernstein D.J., Buchmann J., Dahmen E. (eds) Post-Quantum Cryptography. Springer, Berlin, Heidelberg, 2009.

[151] Micciancio D., Walter M. Fast lattice point enumeration with minimal overhead. In: Indyk P. (ed.), 26th SODA, ACM-SIAM, 2015, pp. 276–294.

[152] Moody D., Perlner R., Smith-Tone D. An optimal structural attack on the ABC multivariate encryption scheme. PQC, 2014.

[153] Odlyzko A. The rise and fall of knapsack cryptosystems. In: Cryptology and Computational Number Theory, 1990, pp. 75–88.

[154] Overbeck R. A multiple birthday attack on NTRU. Secrypt 2008: Proceedings Of The International Conference On Security And Cryptography, 2008, pp. 237-244.

[155] Patarin J. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. In: Advances in Cryptology — EUROCRYPT'96. EUROCRYPT 1996. LNCS 1070, Springer, Heidelberg, 1996, pp. 33–48.

[156] Patarin J. The oil and vinegar signature scheme. Presented at the Dagstuhl Workshop on Cryptography, 1997.

[157] Patarin J., Courtois N., Goubin L., QUARTZ, 128-bit long digital signatures. In: CT-RSA 2001, LNCS 2020, Springer, Heidelberg, 2001, pp. 282–297.

[158] Patarin J., Goubin L., Courtois N. Improved Algorithms for Isomorphisms of Polynomials. In: Advances in Cryptology — EUROCRYPT'98. EUROCRYPT 1998. LNCS 1403. Springer, Berlin, Heidelberg, 1998, pp. 184–200.

[159] Peikert C. A Decade of Lattice Cryptography. Foundations and Trends in Theoretical Computer Science, 10(4), 2016, pp. 283–424.

[160] Perret L. Grobner basis techniques in post-quantum cryptography. Avail. at: https://tel.archives-ouvertes.fr/tel-01417808/document, 2016.

[161] Petzoldt A., Chen M.-Sh., Yang B.-Y., Tao Ch., Ding J. Design Principles for HFEv-based Multivariate Signature Schemes. In: Advances in Cryptology – ASIACRYPT 2015. ASIACRYPT 2015. LNCS 9452. Springer, Berlin, Heidelberg, 2015, pp. 311–334.

[162] Petzoldt A., Ding J., Wang L.C. Eliminating decryption failures from the simple matrix encryption scheme. Cryptology ePrint Archive, Report 2016/010, 2016. Avail. at: http://eprint.iacr.org/2016/010.

[163] Peikert C. An Efficient and Parallel Gaussian Sampler for Lattices. In: Advances in Cryptology — CRYPTO 2010. CRYPTO 2010. LNCS 6223. Springer, Berlin, Heidelberg, pp. 80–97. Avail. at: https://web.eecs.umich.edu/~cpeikert/pubs/pargauss.pdf.

[164] Popov S. The Tangle. April 30, 2018. Version 1.4.3. Avail. at: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf.

[165] Quantum-Safe Cryptography (QSC); Quantum-safe algorithmic framework. ETSI GR QSC 001 V1.1.1 (2016-07), 2016. Avail. at: http://www.etsi.org/standards-search.

[166] Ramachandran N. et al. (?). A Practical Approach To Curbing Address Reuse in the IOTA Protocol. Version: 2018.06. Private communication.

[167] Regev O. On lattices, learning with errors, random linear codes, and cryptography. In: Proc. 37th ACM Symp. on Theory of Computing (STOC), 2005, pp. 84—93.

[168] Reyzin L., Reyzin N. Better than BiBa: Short one-time signatures with fast signing and verifying. In: Information Security and Privacy. ACISP 2002. LNCS 2384. Springer, Berlin, Heidelberg, pp. 144–153. Avail. at: https://www.cs.bu.edu/~reyzin/papers/one-time-sigs.pdf.

[169] Rogaway Ph., Bellare M., Black J., Krovetz T. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. In: ACM Conference on Computer and Communications Security, 2001, pp. 196–205.

[170] Rompel J. One-way functions are necessary and sufficient for secure signatures. STOC'90 Proceedings, 1990, pp. 387–394. Avail. at: https://www.cs.princeton.edu/courses/archive/spr08/cos598D/Rompel.pdf.

[171] Schanck J. Practical Lattice Cryptosystems: NTRUEncrypt and NTRUMLS. Master's thesis, University of Waterloo, 2015.

[172] Shallit J., Frandsen G., Buss J. The Computational Complexity of some Problems of Linear Algebra. Annual Symposium on Theoretical Aspects of Computer Science (STACS 97), 1997, pp. 451–462.

[173] Shim K.-A., Park Ch.-M., Kim A. HiMQ-3: A high speed signature scheme based on multivariate quadratic equations. NIST competition submission. Avail. at: https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[174] Steinfeld R., Albrecht M., Postlethwaite E., Virdia F. Official comments — cfpkm. PQC-forum, 2018. Avail at: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/CFPKM-official-comment.pdf.

[175] Stehlé D., Steinfeld R. Making NTRU as secure as worst-case problems over ideal lattices. In: Advances in Cryptology — EUROCRYPT 2011. EUROCRYPT 2011. LNCS 6632. Springer, Berlin, Heidelberg, 2011, pp. 27–47.

[176] Szepieniec A., Reyhanitabar R., Preneel B. Key Encapsulation from Noisy Key Agreement in the Quantum Random Oracle Model. Cryptology ePrint Archive, Report 2018/884. Avail at: https://eprint.iacr.org/2018/884.pdf.

[177] Tao Ch., Xiang H., Petzoldt A., Ding J. Simple Matrix — A multivariate public key cryptosystem (MPKC) for encryption. Finite Fields and their Applications, vol. 35, 2015, pp. 352–368.

[178] Tao C., Diene A., Tang S., Ding J. Simple Matrix Scheme for Encryption. In: Post-Quantum Cryptography. PQCrypto 2013. LNCS 7932, Springer, Berlin, Heidelberg, 2013, pp. 231–242.

[179] Troika: a ternary hash function. Reference document. Version 1.0.1. Avail. at: https://www.cyber-crypt.com/wp-content/uploads/2018/12/20181221.iota_.troika-reference.v1.0.1.pdf.

[180] van Vredendaal C. Reduced Memory Meet-in-the-Middle Attack against the NTRU Private Key. LMS Journal of Computation and Mathematics, 19(A), 2016, pp. 43–57.

[181] Wetzels J., Bokslag W. Sponges and engines an introduction to Keccak and Keyak. ArXiv preprint, 2015. Avail. at: https://arxiv.org/pdf/1510.02856v2.pdf

[182] Wolf Ch. Multivariate Quadratic Polynomials in Public Key Cryptography. Cryptology ePrint Archive, Report 2005/393, 2005. Avail. at: https://eprint.iacr.org/2005/393.

[183] Wunderer Th. On the Security of Lattice-Based Cryptography Against Lattice Reduction and Hybrid Attacks. Technische Universitat, Darmstadt, Ph.D. Thesis, 2018. Avail at: https://tuprints.ulb.tu-darmstadt.de/8082/1/Dissertation_Wunderer.pdf.

[184] Wunderer Th. Revisiting the hybrid attack: Improved analysis and refined security estimates. Cryptology ePrint Archive, Report 2016/733, 2016. Avail at: http://eprint.iacr.org/2016/733.

[185] Yalcin T., Kavun E. On the Implementation Aspects of Sponge-based Authenticated Encryption for Pervasive Devices. Proceeding CARDIS'12, Proceedings of the 11-th international conference on Smart Card Research and Advanced Applications, 2012, pp. 141–157.

[186] Zhandry M. A note on the quantum collision and set equality problems. Quantum Information & Computation, 15(7&8), 2015, pp. 557–567.