Name of Proposal:

Rainbow

website: www.pqcrainbow.org

Principal Submitter:

Jintai Ding

email: jintai.ding@gmail.com
phone: +1 513 – 394 - 1579
organization: Department of Mathematical Sciences, University of Cincinnati
postal address: French Hall West, OH 45221-0025 Cincinnati, USA

Auxiliary Submitters:
Ming-Shing Chen, Matthias Kannwischer, Jacques Patarin, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang

Inventors: c.f. Submitters

Owners: c.f. Submitters

Jintai Ding (Signature)

Additional Point of Contact:
Bo-Yin Yang
email: by@crypto.tw
phone: 886-2-2788-3799
Fax: 886-2-2782-4814
organization: Academia Sinica
postal address: 128 Academia Road, Section 2

Nankang, Taipei 11529, Taiwan

Rainbow - Algorithm Specification and Documentation

The 3rd Round Proposal

Type: Signature scheme

Family: Multivariate Cryptography, SingleField schemes

1 Changes to the second round submission

We applied the following changes to our submission.

1. **Additional Submitters**: We added two new members to our team: Matthias Kannwischer and Jacques Patarin.

2. **Website**: We created a website for the rainbow signature scheme (www.pqcrainbow.org), where you can find the current submission data (this file), the most recent software implementation, as well as a list of references related to our scheme.

3. **Name Change of the Rainbow variants**: We decided to change the name of the Rainbow variant cyclicRainbow (c.f. 2nd round submission) into CZ-Rainbow (circumzenithal Rainbow). In meteorology, a circumzenithal Rainbow is an upside down Rainbow. By using this name, we want to emphasize that CZ-Rainbow inverts the usual key generation of the Rainbow scheme. We further want to emphasize that CZ-Rainbow does not use any cyclic structure at all.

4. **Parameter Choice**:
   For simplicity, we drop the letters in the parameter proposals and denote the three parameter proposals simply by I, III and V (according to the three security categories). Furthermore, due to some refinements in the complexity analysis of the Rainbow Band Separation and improvements in the MinRank attack (see Sections 9.3, 9.6 and our post in the PQ forum [31]), we adapted the proposed parameters slightly. We therefore have

   - **I**: $\mathbb{F}$=GF(16), $(v_1, o_1, o_2) = (36, 32, 32)$ for the NIST security categories I and II,
   - **III**: $\mathbb{F}$= GF(256), $(v_1, o_1, o_2) = (68, 32, 48)$ for the NIST security categories III and IV, and
   - **V**: $\mathbb{F}$=GF(256), $(v_1, o_1, o_2) = (96, 36, 64)$ for the NIST security category V.

2 History

In this section we give a short overview of the historical background of multivariate public key cryptography and the Rainbow signature scheme.

2.1 Multivariate Public Key Cryptography

For a multivariate public key cryptosystem (MPKC), the public key is given by a set of nonlinear multivariate polynomials over a finite field. The idea of multivariate cryptography started in the 1980's and many leading cryptographers (Ong, Schnorr, Matsumoto, Imai, Harashima, Diffie, Fell, Miyagawa, Tsujii,

Kurosawa, Fujioka and others) tried to build various types of MPKCs. However the Linearization Equations attack of Patarin against the Matsumoto-Imai cryptosystem [15] provided the major impetus for the development of MPKCs.

In general, the public key of a multivariate public key cryptosystem (MPKC) is a system of multivariate quadratic polynomials:

$$\begin{aligned}
p^{(1)}(x_1,\ldots,x_n) &= \sum_{i=1}^{n}\sum_{j=i}^{n} p_{ij}^{(1)}\cdot x_i x_j \quad + \quad \sum_{i=1}^{n} p_i^{(1)}\cdot x_i + p_0^{(1)} \\
p^{(2)}(x_1,\ldots,x_n) &= \sum_{i=1}^{n}\sum_{j=i}^{n} p_{ij}^{(2)}\cdot x_i x_j \quad + \quad \sum_{i=1}^{n} p_i^{(2)}\cdot x_i + p_0^{(2)} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
p^{(m)}(x_1,\ldots,x_n) &= \sum_{i=1}^{n}\sum_{j=i}^{n} p_{ij}^{(m)}\cdot x_i x_j \quad + \quad \sum_{i=1}^{n} p_i^{(m)}\cdot x_i + p_0^{(m)}
\end{aligned}$$

Here, all coefficients and variables are from $\mathbb{F}_q$, a finite field with q elements. The set of public polynomials

$$\mathcal{P}(x_1,\ldots,x_n) = \left(p^{(1)}(x_1,\ldots,x_n),\ldots,p^{(m)}(x_1,\ldots,x_n)\right),$$

is mathematically a map from $\mathbb{F}_q^n$ to $\mathbb{F}_q^m$.

The public operations of encrypting a message or verifying a signature correspond to simply evaluating $\mathcal{P}(x_1,\ldots,x_n)$. The process of decrypting a ciphertext or generating a signature corresponds to an "inversion" of the map $\mathcal{P}(x_1,\ldots,x_n)$. This is equivalent to solving an instance of the following problem:

MQ Problem: Given a system $\mathcal{P}$ of m multivariate quadratic polynomials $p^{(1)}(\mathbf{x}),\ldots,p^{(m)}(\mathbf{x})$ in n variables $x_1,\ldots,x_n$ as shown above, find a vector $\bar{\mathbf{x}} = (\bar{x}_1,\ldots,\bar{x}_n)$ such that

$$\mathcal{P}(\bar{x}_1,\ldots,\bar{x}_n) = (0,\ldots,0),$$

or

$$p^{(1)}(\bar{\mathbf{x}}) = \cdots = p^{(m)}(\bar{\mathbf{x}}) = 0.$$

The general MQ problem is proved to be NP-hard on every finite field GF(q). The proof is particularly simple and direct for $q = 2$ (cf. [16]). The most difficult instances of MQ are generally obtained when m and n are of the same order of magnitude (when m is much larger than n, or when n is much larger than m some efficient algorithms are known). It is also interesting to note that very often the best known algorithms on MQ have a similar complexity for worse cases than for random cases. This is also true for quantum algorithms. At present, there does not exist any quantum algorithm algorithm that can solve this MQ problem efficiently, and since MQ is NP-hard we have an argument

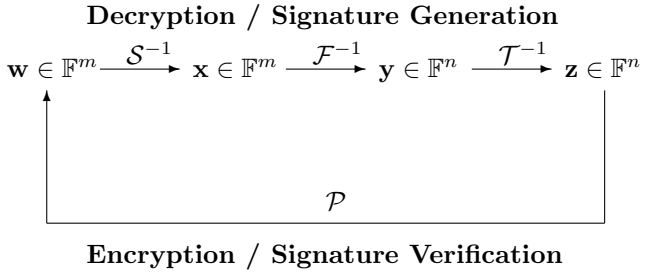


Figure 1: General work flow for multivariate schemes

to expect that this will still be the case in the future (for more information on quantum algorithms solving the MQ Problem c.f. Section 9.8).

In order to enable efficient decryption and signature generation, it is necessary to build a secret trapdoor for the map $\mathcal{P}(x_1, \ldots, x_n)$.

For the standard construction of a multivariate public key cryptosystem, one chooses a system $\mathcal{F}$ of m quadratic polynomials in n variables, which can be easily inverted (central map). After that, one chooses two affine invertible maps $\mathcal{S}$ and $\mathcal{T}$ to hide the structure of the central map $\mathcal{F}$ in the public key. The public key of the cryptosystem is the composed quadratic map $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ which is supposed to be difficult to invert. The private key consists of $\mathcal{S}$, $\mathcal{F}$ and $\mathcal{T}$ and therefore allows one to invert $\mathcal{P}$. The maps $\mathcal{S}$ and $\mathcal{T}$ are used to protect the map $\mathcal{F}$.

The standard process for encryption and decryption or signature generation and verification works as follows (see Figure 1):

Encryption Schemes ($m \geq n$)

Encryption: To encrypt a message $\mathbf{z} \in \mathbb{F}^n$, one simply computes $\mathbf{w} = \mathcal{P}(\mathbf{z})$. The ciphertext of the message $\mathbf{z}$ is $\mathbf{w} \in \mathbb{F}^m$.

Decryption: To decrypt the ciphertext $\mathbf{w} \in \mathbb{F}^m$, one computes sequentially $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{w})$, $\mathbf{y} = \mathcal{F}^{-1}(\mathbf{x})$ and $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$. $\mathbf{z} \in \mathbb{F}^n$ is the plaintext corresponding to the ciphertext $\mathbf{w}$. Since $m \geq n$, the pre-image of $\mathbf{x}$ under $\mathcal{F}$ and therefore the resulting plaintext is unique.

Signature Schemes ($m \leq n$)

Signature Generation: To sign a document d, we use a hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{F}^m$ to compute the value $\mathbf{w} = \mathcal{H}(d) \in \mathbb{F}^m$. Then we compute $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{h})$, $\mathbf{y} = \mathcal{F}^{-1}(\mathbf{x})$ and $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$. The signature of the document d is $\mathbf{z} \in \mathbb{F}^n$. Here, $\mathcal{F}^{-1}(\mathbf{x})$ means finding one (of the possibly many) pre-images of

$\mathbf{x}$ under the central map $\mathcal{F}$. Since $n \geq m$ we can be sure that such a pre-image exists. Therefore every message has a signature.

Verification: To verify the authenticity of a signature $\mathbf{z}$, one simply computes $\mathbf{w}' = \mathcal{P}(\mathbf{z})$ and the hash value $\mathbf{w} = \mathcal{H}(d)$ of the document. If $\mathbf{w}' = \mathbf{w}$ holds, the signature is accepted, otherwise rejected.

2.2 Rainbow history

The history of the Rainbow signature scheme can be traced back to Patarin's Linearization Equations attack against the Matsumoto-Imai cryptosystem of 1995 [22]. One year later, in 1996, Patarin derived from this attack the Oil and Vinegar signature scheme (OV) [23]. After the balanced version of this scheme was broken by an invariant subspace attack in 1998 [19], Kipnis, Patarin and Goubin proposed in 1999 the Unbalanced Oil and Vinegar Signature Scheme (UOV) [18] which is still believed to be secure. To enhance the efficiency of this scheme, Ding and Schmidt proposed in 2005 the Rainbow signature scheme [11], which can be seen as a multi layered version of UOV.

The Rainbow signature scheme [11] with u layers can be described as follows. Let $\mathbb{F}_q$ be a finite field with q elements and $v_1 < v_2 < \cdots < v_u < v_{u+1} = n$ be integers. We set $V_i = \{1, \ldots, v_i\}$, $o_i = v_{i+1} - v_i$ and $O_i = \{v_i, \ldots, v_{i+1}\}$ $(i = 1, \ldots, u)$. Therefore we obtain $|V_i| = v_i$ and $|O_i| = o_i$ $(i = 1, \ldots, u)$.

The central map $\mathcal{F}$ of Rainbow consists of $m = n - v_1$ multivariate quadratic polynomials $f^{(v_1+1)}, \ldots, f^{(n)}$ of the form

$$f^{(k)}(\mathbf{x}) = \sum_{i,j \in V_\ell} \alpha_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell, j \in O_\ell} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell \cup O_\ell} \gamma_i^{(k)} x_i + \eta^{(k)}, \qquad (1)$$

where $\ell \in \{1, \ldots, u\}$ is the only integer such that $k \in O_\ell$.

Note that, in every polynomial $f^{(k)}$ with $k \in O_\ell$, there is no quadratic term $x_i x_j$ where both i and j are in O_ℓ. This fact will later be used for generating the signature. Such polynomials were called Oil–Vinegar polynomials when the OV schemes was proposed.

To hide the structure of $\mathcal{F}$ in the public key, one composes it with two invertible affine or linear maps $\mathcal{S} : \mathbb{F}^m \to \mathbb{F}^m$ and $\mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$. Hence, the *public key* of Rainbow has the form $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T} : \mathbb{F}^n \to \mathbb{F}^m$, the *private key* consists of the three maps $\mathcal{S}$, $\mathcal{F}$ and $\mathcal{T}$ and therefore allows the inversion of the public key map.

Signature Generation: To generate a signature for a message (or hashvalue) $\mathbf{w} \in \mathbb{F}^m$, one performs the following three steps.

1. Compute $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{w}) \in \mathbb{F}^m$.
2. Compute a pre-image $\mathbf{y}$ of $\mathbf{x}$ under the central map $\mathcal{F}$ using the central map inversion algorithm below.
3. Compute the signature $\mathbf{z} \in \mathbb{F}^n$ by $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$.

Algorithm Inversion of the Rainbow central map

Input: Rainbow central map $\mathcal{F} = (f^{(v_1+1)}, \ldots, f^{(n)})$, vector $\mathbf{x} \in \mathbb{F}^m$.
Output: vector $\mathbf{y} \in \mathbb{F}^n$ with $\mathcal{F}(\mathbf{y}) = \mathbf{x}$.

1: Choose random values for the variables $y_1, \ldots, y_{v_1}$ and substitute these values into the polynomials $f^{(i)}$ $(i = v_1 + 1, \ldots, n)$.
2: **for** $\ell = 1$ to u **do**
3: Perform Gaussian Elimination on the polynomials $f^{(i)}$ $(i \in O_\ell)$ to get the values of the variables y_i $(i \in O_\ell)$.
4: Substitute the values of y_i $(i \in O_\ell)$ into the polynomials $f^{(i)}$ $(i = v_{\ell+1} + 1, \ldots, n)$.
5: **end for**
6: **return** $\mathbf{y} = (y_1, \ldots, y_n)$

Signature Verification: To check, if $\mathbf{z} \in \mathbb{F}^n$ is a valid signature for a message $\mathbf{w} \in \mathbb{F}^m$, one simply computes $\mathbf{w}' = \mathcal{P}(\mathbf{z})$. If $\mathbf{w}' = \mathbf{w}$ holds, the signature is accepted, otherwise rejected.

The current submission basically describes the Rainbow signature scheme as it was proposed in the original paper [11] of 2005 with two layers of OV systems (i.e. $u = 2$). We only adopt a small number of modifications.

- Due to some refined attacks (c.f. Sections 9.3 and 9.6), we adapt the parameters of Rainbow slightly.
- We offer a variant of Rainbow with reduced public key size based on techniques from [27, 28], which we call now circumzenithal (CZ) Rainbow and a compressed variant which additionally reduces the size of the private key.
- To speed up the key generation process, we use a new technique proposed by Petzoldt in [28].

3 Algorithm Specification

In this section we present the Rainbow signature scheme as proposed in [11].

3.1 Parameters

- finite field $\mathbb{F} = \mathbb{F}_q$ with q elements
- integers $0 < v_1 < \cdots < v_u < v_{u+1} = n$
- index sets $V_i = \{1, \ldots, v_i\}$, $O_i = \{v_i + 1, \ldots, v_{i+1}\}$ $(i = 1, \ldots, u)$
  Note that each $k \in \{v_1 + 1, \ldots, n\}$ is contained in exactly one of the sets O_i.
- we have $|V_i| = v_i$ and set $o_i := |O_i|$ $(i = 1, \ldots, u)$

- number of equations: $m = n - v_1$
- number of variables: n

3.2 Key Generation

Private Key. The *private key* consists of

- two invertible affine maps $\mathcal{S} : \mathbb{F}^m \to \mathbb{F}^m$ and $\mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$
- the quadratic *central* map $\mathcal{F} : \mathbb{F}^n \to \mathbb{F}^n$, consisting of m multivariate polynomials $f^{(v_1+1)}, \ldots, f^{(n)}$.
  Remember that, according to the definition of the sets O_i (see Section 3.1), there exists, for any $k \in \{v_1 + 1, \ldots, n\}$ exactly one $\ell \in \{1, \ldots, u\}$ such that $k \in O_\ell$. The polynomials $f^{(k)}$ $(k = v_1+1, \ldots, n)$ are of the form

$$\begin{aligned} f^{(k)}(x_1, \ldots, x_n) &= \sum_{i,j \in V_\ell, i \le j} \alpha_{ij}^{(k)} x_i x_j \\ &+ \sum_{i \in V_\ell, j \in O_\ell} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell \cup O_\ell} \gamma_i^{(k)} x_i + \delta^{(k)}, \quad (2) \end{aligned}$$

  where $\ell \in \{1, \ldots, u\}$ is the only integer such that $k \in O_\ell$ (see above). The coefficients $\alpha_{ij}^{(k)}$, $\beta_{ij}^{(k)}$, $\gamma_i^{(k)}$ and $\delta^{(k)}$ are randomly chosen elements of $\mathbb{F}$.

The size of the private key is

$$\underbrace{m \cdot (m+1)}_{\text{affine map } \mathcal{S}} + \underbrace{n \cdot (n+1)}_{\text{affine map } \mathcal{T}} + \underbrace{\sum_{i=1}^{u} \left(\frac{v_i \cdot (v_i+1)}{2} + v_i \cdot o_i + v_i + o_i + 1 \right)}_{\text{central map } \mathcal{F}}$$

field elements.

Public Key. The *public key* is the composed map

$$\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T} : \mathbb{F}^n \to \mathbb{F}^m$$

and therefore consists of m quadratic polynomials in the ring $\mathbb{F}[x_1, \ldots, x_n]$.

The size of the public key is

$$m \cdot \frac{(n+1) \cdot (n+2)}{2}$$

field elements.

3.3 Signature Generation

Given a document d to be signed, one uses a hash function $\mathcal{H} : \{0,1\} \to \mathbb{F}^m$ to compute the hash value $\mathbf{h} = \mathcal{H}(d) \in \mathbb{F}^m$. A signature $\mathbf{z} \in \mathbb{F}^n$ of the document d is then computed as follows.

1. Compute $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{h}) \in \mathbb{F}^m$.
2. Compute a pre-image $\mathbf{y} \in \mathbb{F}^n$ of $\mathbf{x}$ under the central map $\mathcal{F}$. This is done as shown in Algorithm 1.
3. Compute the signature $\mathbf{z} \in \mathbb{F}^n$ by $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$.

Algorithm 1 Inversion of the Rainbow central map

Input: Rainbow central map $\mathcal{F} = (f^{(v_1+1)}, \ldots, f^{(n)})$, vector $\mathbf{x} \in \mathbb{F}^m$.
Output: vector $\mathbf{y} \in \mathbb{F}^n$ with $\mathcal{F}(\mathbf{y}) = \mathbf{x}$.

1: Choose random values for the variables $y_1, \ldots, y_{v_1}$ and substitute these values into the polynomials $f^{(i)}$ $(i = v_1 + 1, \ldots, n)$.
2: **for** $\ell = 1$ to u **do**
3: Perform Gaussian Elimination on the polynomials $f^{(i)}$ $(i \in O_\ell)$ to get the values of the variables y_i $(i \in O_\ell)$.
4: Substitute the values of y_i $(i \in O_\ell)$ into the polynomials $f^{(i)}$ $(i = v_{\ell+1} + 1, \ldots, n)$.
5: **end for**
6: **return** $\mathbf{y} = (y_1, \ldots, y_n)$

3.4 Signature Verification

Given a document d and a signature $\mathbf{z} \in \mathbb{F}^n$, the authenticity of the signature is checked as follows:

1. Use the hash function $\mathcal{H}$ to compute the hash value $\mathbf{h} = \mathcal{H}(d) \in \mathbb{F}^m$.
2. Compute $\mathbf{h}' = \mathcal{P}(\mathbf{z}) \in \mathbb{F}^m$.

If $\mathbf{h}' = \mathbf{h}$ holds, the signature $\mathbf{z}$ is accepted, otherwise it is rejected.

The Rainbow signature scheme can be defined for any number of layers u. For $u = 1$ we obtain the well known UOV signature scheme [18]. However, choosing $u = 2$ leads to a scheme with more efficient computations and smaller key sizes at the same level of security. Choosing $u > 2$ gives only a very small benefit in terms of performance, but needs larger keys to reach the same security level. Therefore, for ease of implementation, 2 is a common choice for the number of Rainbow layers.

The following algorithms `RainbowKeyGen`, `RainbowSign` and `RainbowVer` illustrate the key generation, signature generation and signature verification processes of Rainbow in algorithmic form.

Note that, in order to speed up the key generation of our scheme, we developed for our implementation alternative key generation algorithms. The details of these algorithms are described in Section 4 (see Algorithms 9 and 10).

Algorithm 2 `RainbowKeyGen`: Key Generation of Rainbow

Input: Rainbow parameters (q, v_1, o_1, o_2)
Output: Rainbow key pair (sk, pk)

1: $m \leftarrow o_1 + o_2$
2: $n \leftarrow m + v_1$
3: **repeat**
4: $\quad M_S \leftarrow \texttt{Matrix}(q, m, m)$
5: **until** IsInvertible(M_S) == **TRUE**
6: $c_S \leftarrow_R \mathbb{F}^m$
7: $\mathcal{S} \leftarrow \texttt{Aff}(M_S, c_S)$
8: $InvS \leftarrow M_S^{-1}$
9: **repeat**
10: $\quad M_T \leftarrow \texttt{Matrix}(q, n, n)$
11: **until** IsInvertible(M_T) == **TRUE**
12: $c_T \leftarrow_R \mathbb{F}^n$
13: $\mathcal{T} \leftarrow \texttt{Aff}(M_T, c_T)$
14: $InvT \leftarrow M_T^{-1}$
15: $\mathcal{F} \leftarrow \texttt{Rainbowmap}(q, v_1, o_1, o_2)$
16: $\mathcal{P} \leftarrow \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$
17: sk $\leftarrow (InvS, c_S, \mathcal{F}, InvT, c_T)$
18: pk $\leftarrow \mathcal{P}$
19: **return** (sk, pk)

The possible input values of Algorithm `RainbowKeyGen` are specified in Section 3.8. $\texttt{Matrix}(q, m, n)$ returns an $m \times n$ matrix with coefficients chosen uniformly at random in $\mathbb{F}_q$, while $\texttt{Aff}(M, c)$ returns the affine map $M \cdot x + c$.

$\texttt{Rainbowmap}(q, v_1, o_1, o_2)$ returns a Rainbow central map according to the parameters (q, v_1, o_1, o_2) (see equation (2)). The coefficients $\alpha_{ij}^{(k)}$, $\beta_{ij}^{(k)}$, $\gamma_i^{(k)}$ and $\delta^{(k)}$ are hereby chosen uniformly at random from $\mathbb{F}_q$.

Altogether, we need in `RainbowKeyGen` the following number of randomly chosen $\mathbb{F}_q$-elements:

$$\underbrace{m \cdot (m+1)}_{\text{affine map } \mathcal{S}} + \underbrace{n \cdot (n+1)}_{\text{affine map } \mathcal{T}} + \underbrace{o_1 \cdot \left(\frac{v_1 \cdot (v_1+1)}{2} + v_1 \cdot o_1 + v_1 + o_1 + 1 \right)}_{\text{central polynomials of the first layer}}$$
$$+ \underbrace{o_2 \cdot \left(\frac{(v_1+o_1) \cdot (v_1+o_1+1)}{2} + (v_1+o_1) \cdot o_2 + v_1 + o_1 + o_2 + 1 \right)}_{\text{central polynomials of the second layer}}$$

Algorithm 3 `RainbowSign`: Signature generation process of Rainbow

Input: Rainbow private key $(InvS, c_S, \mathcal{F}, InvT, c_T)$, document d
Output: signature $\mathbf{z} \in \mathbb{F}^n$ such that $\mathcal{P}(\mathbf{z}) = \mathcal{H}(d)$

1: $\mathbf{h} \leftarrow \mathcal{H}(d)$
2: $\mathbf{x} \leftarrow InvS \cdot (\mathbf{h} - c_S)$
3: $\mathbf{y} \leftarrow \texttt{InvF}(\mathcal{F}, \mathbf{x})$
4: $\mathbf{z} \leftarrow InvT \cdot (\mathbf{y} - c_T)$
5: **return z**

Algorithm 4 `InvF`: Inversion of the Rainbow central map

Input: Rainbow central map $\mathcal{F} = (f^{(v_1+1)}, \ldots, f^{(n)})$, vector $\mathbf{x} \in \mathbb{F}^m$.
Output: vector $\mathbf{y} \in \mathbb{F}^n$ with $\mathcal{F}(\mathbf{y}) = \mathbf{x}$.

1: **repeat**
2: $\quad y_1, \ldots, y_{v_1} \leftarrow_R \mathbb{F}$
3: $\quad \hat{f}^{(v_1+1)}, \ldots, \hat{f}^{(n)} \leftarrow f^{(v_1+1)}(y_1, \ldots, y_{v_1}), \ldots, f^{(n)}(y_1, \ldots, y_{v_1})$.
4: $\quad t, (y_{v_1+1}, \ldots, y_{v_2}) \leftarrow \texttt{Gauss}(\hat{f}^{(v_1+1)} = x_{v_1+1}, \ldots, \hat{f}^{(v_2)} = x_{v_2})$
5: $\quad$ **if** $t ==$ **TRUE then**
6: $\quad\quad \hat{f}^{(v_2+1)}, \ldots, \hat{f}^{(n)} \leftarrow \hat{f}^{(v_2+1)}(y_{v_1+1}, \ldots, y_{v_2}), \ldots, \hat{f}^{(n)}(y_{v_1+1}, \ldots, y_{v_2})$
7: $\quad\quad t, (y_{v_2+1}, \ldots, y_n) \leftarrow \texttt{Gauss}(\hat{f}^{(v_2+1)} = x_{v_2+1}, \ldots, \hat{f}^{(n)} = x_n)$
8: $\quad$ **end if**
9: **until** $t ==$ **TRUE**
10: **return** $\mathbf{y} = (y_1, \ldots, y_n)$

In Algorithm `InvF`, the function `Gauss` returns a binary value $t \in \{$**TRUE**, **FALSE**$\}$ indicating whether the given linear system is solvable, and if so a random solution of the system. In `InvF` we make use of at least v_1 random field elements (depending on how often we have to perform the loop to find a solution).

Algorithm 5 `RainbowVer`: Signature verification of Rainbow

Input: document d, signature $\mathbf{z} \in \mathbb{F}^n$
Output: **TRUE** or **FALSE**

1: $\mathbf{h} \leftarrow \mathcal{H}(d)$
2: $\mathbf{h}' \leftarrow \mathcal{P}(\mathbf{z})$
3: **if h'==h then**
4: $\quad$ **return TRUE**
5: **else**
6: $\quad$ **return FALSE**
7: **end if**

3.5 Changes needed to achieve EUF-CMA Security

The standard Rainbow signature scheme as described above provides only universal unforgeability. In order to obtain EUF-CMA security, we apply a transformation similar to that in [29]. The main difference is the use of a random binary vector r called salt. Instead of generating a signature for $\mathbf{h} = \mathcal{H}(d)$ as in Algorithm `RainbowSign`, we generate a signature for $\mathcal{H}(\mathcal{H}(d)||r)$. The modified signature has the form $\sigma = (\mathbf{z}, r)$, where $\mathbf{z}$ is a standard Rainbow signature. By doing so, we ensure that an attacker is not able to forge any (hash value/ signature) pair. In particular, we apply the following changes to the algorithms `RainbowKeyGen`, `RainbowSign` and `RainbowVer`.

- In the algorithm `RainbowKeyGen`*, we choose an integer ℓ as the length of a random salt; ℓ is appended both to the private and the public key.
- In the algorithm `RainbowSign`*, we choose first randomly the values of the vinegar variables $\in F^{v_1}$; after that, we choose a random salt $r \in \{0,1\}^\ell$ and perform the standard Rainbow signature generation process for $\mathbf{h} = \mathcal{H}(\mathcal{H}(d)||r)$ to obtain a signature $\sigma = (\mathbf{z}||r)$. If the linear system in step 2 of the signature generation process has no solutions, we choose a new value for the salt r and try again.
- The verification algorithm `RainbowVer`* returns **TRUE** if $\mathcal{P}(\mathbf{z}) = \mathcal{H}(\mathcal{H}(d)||r)$, and **FALSE** otherwise

Algorithms `RainbowKeyGen`*, `RainbowSign`* and `RainbowVer`* show the modified key generation, signing and verification algorithms.

Algorithm 6 `KeyGen`*: Modified Key Generation Algorithm for Rainbow

Input: Rainbow parameters (q, v_1, o_1, o_2), length ℓ of salt
Output: Rainbow key pair (sk, pk)

1: $pk, sk \leftarrow$ `RainbowKeyGen`(q, v_1, o_1, o_2)
2: $sk \leftarrow sk, \ell$
3: $pk \leftarrow pk, \ell$
4: **return** (sk, pk)

Algorithm 7 `RainbowSign`*: Modified signature generation process for Rainbow

Input: document d, Rainbow private key $(InvS, c_S, \mathcal{F}, InvT, c_T)$, length ℓ of the salt

Output: signature $\sigma = (\mathbf{z}, r) \in \mathbb{F}^n \times \{0,1\}^\ell$ such that $\mathcal{P}(\mathbf{z}) = \mathcal{H}(\mathcal{H}(d)||r)$

1: **repeat**
2: $y_1, \ldots, y_{v_1} \leftarrow_R \mathbb{F}$
3: $\hat{f}^{(v_1+1)}, \ldots, \hat{f}^{(n)} \leftarrow f^{(v_1+1)}(y_1, \ldots, y_{v_1}), \ldots, f^{(n)}(y_1, \ldots, y_{v_1})$
4: $(\hat{F}, c_F) \leftarrow \texttt{Aff}^{-1}(\hat{f}^{(v_1+1)}, \ldots, \hat{f}^{(n)})$
5: **until** IsInvertible$(\hat{F})$ == **TRUE**
6: $InvF = \hat{F}^{-1}$
7: **repeat**
8: $r \leftarrow \{0,1\}^\ell$
9: $\mathbf{h} \leftarrow \mathcal{H}(\mathcal{H}(d)||r)$
10: $\mathbf{x} \leftarrow InvS \cdot (\mathbf{h} - c_S)$
11: $(y_{v_1+1}, \ldots, y_{v_2}) \leftarrow InvF \cdot ((x_{v_1+1}, \ldots, x_{v_2}) - c_F)$
12: $\hat{f}^{(v_2+1)}, \ldots, \hat{f}^{(n)} \leftarrow \hat{f}^{(v_2+1)}(y_{v_1+1}, \ldots, y_{v_2}), \ldots, \hat{f}^{(n)}(y_{v_1+1}, \ldots, y_{v_2})$
13: $t, (y_{v_2+1}, \ldots, y_n) \leftarrow \text{Gauss}(\hat{f}^{(v_2+1)} = x_{v_2+1}, \ldots, \hat{f}^{(n)} = x_n)$
14: **until** t == **TRUE**
15: $\mathbf{z} = InvT \cdot (\mathbf{y} - c_T)$
16: $\sigma \leftarrow (\mathbf{z}, r)$
17: **return** σ

In Algorithm `RainbowSign`*, the function `Aff`$^{-1}$ takes as input an affine map $\mathcal{G} = M \cdot x + c$ and returns M and c. Note that, in line 9 of the algorithm, we do not compute $\mathcal{H}(d||r)$, but $\mathcal{H}(\mathcal{H}(d)||r)$. In case we have to perform this step several times for a long message d, this improves the efficiency of our scheme significantly.

Algorithm 8 `RainbowVer`*: Modified signature verification process for Rainbow

Input: document d, signature $\sigma = (\mathbf{z}, r) \in \mathbb{F}^n \times \{0,1\}^\ell$

Output: boolean value **TRUE** or **FALSE**

1: $\mathbf{h} \leftarrow \mathcal{H}(\mathcal{H}(d)||r)$
2: $\mathbf{h}' \leftarrow \mathcal{P}(\mathbf{z})$
3: **if** $\mathbf{h}$' == $\mathbf{h}$ **then**
4: **return TRUE**
5: **else**
6: **return FALSE**
7: **end if**

Similar to [29] we find that every attacker, who can break the EUF-CMA security of the modified scheme, can also break the standard Rainbow signature scheme.

In order to get a secure scheme, we have to ensure that no salt is used for

more than one signature. Under the assumption of up to 2^{64} signatures being generated with the system [21], we choose the length of the salt r to be 128 bit (independent of the security level).

3.6 Note on the Hash Function

In our implementation we use SHA-2 as the underlying hash function. The SHA-2 hash function family contains the four hash functions SHA224, SHA256, SHA384 and SHA512 with output lengths of 224, 256, 384 and 512 bits respectively. In the Rainbow instance I aimed at the NIST security categories I and II, we use SHA256 as the underlying hash function. For the Rainbow instances III and V, aimed for the security categories III/IV and V, we use SHA384 and SHA512 respectively.

In case a slightly longer (non standard) hash output is needed for our scheme (Rainbow schemes over GF(256)), we proceed as follows:

In order to obtain a hash value of length $256 < k < 384$ bit for a document d, we set

$$\mathcal{H}(d) = \text{SHA256}(d) \;\; || \;\; \text{SHA256}(\text{SHA256}(d))|_{1,\ldots,k-256}.$$

(analogously for hash values of length $384 < k < 512$ bits and $k > 512$ bits)
By doing so, we ensure that a collision attack against the hash function $\mathcal{H}$ is at least as hard as a collision attack against SHA256 (rsp. SHA384, SHA512).

3.7 Note on the Generation of Random Field Elements

During the key and signature generation of Rainbow, we make use of a large number of random field elements. These are obtained by calling a cryptographic random number generator such as the one from the OpenSSL library. The random number generator used in our implementations is the `AES CTR_DRBG` function. In debug mode, our software can either generate the random bits and store them in a file, or read the required random bits from a file (for Known Answer Tests).

3.8 Parameter Choice

We propose the following three parameter sets for Rainbow

I $(\mathbb{F}, v_1, o_1, o_2) = (\text{GF}(16), 36, 32, 32)$ (64 equations, 100 variables)

III $(\mathbb{F}, v_1, o_1, o_2) = (\text{GF}(256), 68, 32, 48)$ (80 equations, 148 variables)

V $(\mathbb{F}, v_1, o_1, o_2) = (\text{GF}(256), 96, 36, 64)$ (100 equations, 196 variables)

Here, the roman numeral indicates the NIST security category which the given Rainbow instance aims at (see Section 8.2). Note that each of the given parameter sets fulfills the requirements of two of the NIST security categories. In

particular, the Rainbow instance I meets the requirements of the security categories I and II, the parameter set III is designed for security categories III and IV, and the Rainbow instance V meets the requirements of security category V.

For all three parameter sets we propose a standard version as well as two variants with smaller keys (see next section).

3.9 Rainbow Variants

In addition to the standard Rainbow scheme, we propose in this submission a variant of Rainbow called CZ-Rainbow or circumzenithal Rainbow. As the name suggests we reverse the key generation process of standard Rainbow: instead of computing the public key from the private key, we fix major parts of the public key and then compute the central Rainbow map from the public key.

The CZ-Rainbow scheme is based on and inspired by the "CyclicRainbow" signature scheme proposed by Petzoldt et al. in [27]. In CyclicRainbow we allow some cyclic structure inside the Rainbow public key. However, by looking carefully at this basic idea we see that the process of generating a public key from the private key is a partially reversible process. That is, we can actually select randomly parts of the public key and parts of the private key, and then can calculate from this the rest of the public key and the private key. Since this process is reversible, one can see easily that this change in producing the keys does not have any impact on the security.

The CZ-Rainbow scheme is exactly based on this idea of reversing the key generation process of Rainbow. In order to emphasize the fact that CZ-Rainbow reverses the key generation of Rainbow and that it does not use cyclic matrices at all we have decided to use the new name CZ-Rainbow or circumzenithal Rainbow[1].

Since a part of the public key can be randomly selected anyway, the key idea in the CZ-Rainbow scheme is to use a cryptographic PRNG to generate that part of the public key. To create a key pair of CZ-Rainbow, we choose the two linear maps $\mathcal{S}$ and $\mathcal{T}$ at random and create the yellow parts of the public key from a small seed $\mathbf{s}_{\text{pub}}$ using a PRNG. As described in [27] we can from this compute the central map $\mathcal{F}$ of the Rainbow scheme as well as the missing parts of the public key (blue in Figure 2). Therefore, instead of storing the whole public key, we can simply store the seed $\mathbf{s}_{\text{pub}}$ as well as some small portion $\mathcal{P}_2$ of the public key. By doing so, we can reduce the public key size of the Rainbow scheme by a factor of up to 70%.

However, the key generation algorithm of cyclic Rainbow presented in [27] is far too inefficient for our purposes. Therefore, we developed a new key generation algorithm for the CZ-Rainbow scheme (see Section 4.3), which is only slightly slower than the standard key generation process. However, during the verification process, we have to decompress the public key before evaluating it, and the verification process of the scheme is slowed down significantly.[2]

[1] In meteorology, a circumzenithal Rainbow is an upside down Rainbow

[2] However, we want to note that this slowdown is caused completely by the use of the cryptographically secure, AES-based PRNG supplied by OpenSSL (which is the same as the

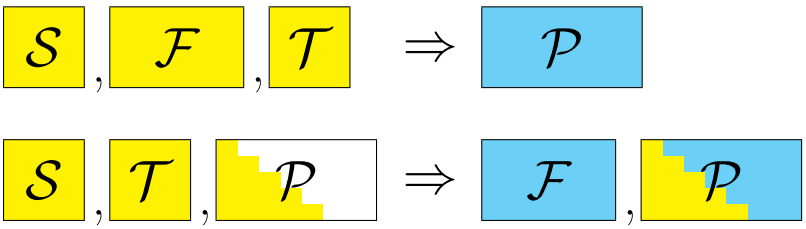


Figure 2: Key generation of standard Rainbow (above) and of CZ-Rainbow. The yellow parts are chosen by the user, the blue parts are computed during the key generation process.

Besides the standard and the CZ variant, we also propose a Rainbow variant with a "compressed" key. The key generation process of this Compressed Rainbow variant works exactly as that of CZ-Rainbow, only that we don't store the computed central map. Instead of this, the private key of Compressed Rainbow consists of the two seeds s_{pub} and s_{priv} which was used to create the two matrices $\mathcal{S}$ and $\mathcal{T}$. During signature generation, we therefore have to run the key generation process again to generate the central map from s_{pub} and s_{priv}. This slows down the signature generation process significantly, but reduces the private key size dramatically.

4 Efficient Key Generation

In this section we describe how to generate Rainbow key pairs in an efficient way. In the first subsection (Subsection 4.1) we introduce some general conventions and notations used in the remainder of this proposal. Subsection 4.2 then describes how to efficiently generate a key pair of the standard Rainbow signature scheme, while Subsection 4.3 deals with the key generation of the CZ-Rainbow signature scheme.

4.1 Conventions

In order to speed up the key generation process, we make the following restrictions on our Rainbow instances.

- We restrict ourselves to Rainbow schemes with two layers. Thus, the scheme is determined by the parameters v_1, o_1 and o_2 and we have $m = o_1 + o_2$ equations and $n = v_1 + m$ variables. Note that our parameter proposals are of this type.

NIST supplied one) to generate the "fixed" parts of the public key. By using a faster stream cipher such as ChaCha this slowdown can be avoided nearly completely.

- We restrict ourselves to homogeneous maps $\mathcal{S}$, $\mathcal{F}$ and $\mathcal{T}$. Note that, due to this choice, the public key $\mathcal{P}$ is a homogeneous quadratic map from $\mathbb{F}^n$ to $\mathbb{F}^m$.
  It is widely accepted that the complexity of direct attacks against a multivariate system $\mathcal{P}$ is determined solely by the homogeneous part of $\mathcal{P}$ of highest degree. All other attacks against the Rainbow scheme neglect the linear part of the Rainbow public key and only use the (symmetric) matrices representing the homogeneous quadratic part. Therefore, restricting to homogeneous keys does not weaken the security of Rainbow.

- We use linear maps $\mathcal{S}$ and $\mathcal{T}$ of the form

$$\begin{aligned} \mathcal{S} &= \begin{pmatrix} I_{o_1 \times o_1} & S'_{o_1 \times o_2} \\ 0_{o_2 \times o_1} & I_{o_2 \times o_2} \end{pmatrix}, \\ \mathcal{T} &= \begin{pmatrix} I_{v_1 \times v_1} & T^{(1)}_{v_1 \times o_1} & T^{(2)}_{v_1 \times o_2} \\ 0_{o_1 \times v_1} & I_{o_1 \times o_1} & T^{(3)}_{o_1 \times o_2} \\ 0_{o_2 \times v_1} & 0_{o_2 \times o_1} & I_{o_2 \times o_2} \end{pmatrix}. \end{aligned} \tag{3}$$

  Note that, for every Rainbow public key $\mathcal{P}$, there exists a corresponding private key $(\mathcal{S}, \mathcal{F}, \mathcal{T})$ with $\mathcal{S}$ and $\mathcal{T}$ being of form (3) (c.f. [33]). Therefore, the previous assumption does not weaken the security of our scheme.
  For our special choice of $\mathcal{S}$ and $\mathcal{T}$ we have $\det(\mathcal{S}) = \det(\mathcal{T}) = 1$ and

$$\begin{aligned} \mathcal{S}^{-1} &= \begin{pmatrix} I_{o_1 \times o_1} & S'_{o_1 \times o_2} \\ 0_{o_2 \times o_1} & I_{o_2 \times o_2} \end{pmatrix} = \mathcal{S}, \\ \mathcal{T}^{-1} &= \begin{pmatrix} I & T^{(1)} & T^{(1)} \cdot T^{(3)} - T^{(2)} \\ 0 & I & T^{(3)} \\ 0 & 0 & I \end{pmatrix}. \end{aligned} \tag{4}$$

  For abbreviation, we set $T^{(4)} := T^{(1)} \cdot T^{(3)} - T^{(2)}$.

Additionally to enabling us to speed up the key generation process, these conventions also reduce the key sizes of Rainbow slightly. Since there are no linear and constant terms any more, the size of the public key is now given as

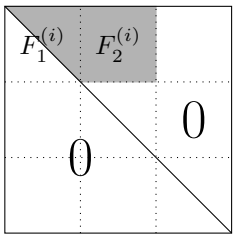
$$\text{size}_{pk} = m \cdot \frac{n \cdot (n+1)}{2}$$

field elements, the size of the private key is

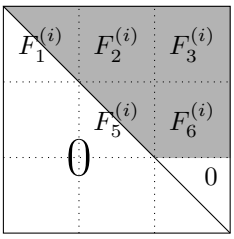
$$\begin{aligned} \text{size}_{sk} &= \underbrace{o_1 o_2}_{\text{linear map } \mathcal{S}} + \underbrace{v_1(o_1 + o_2) + o_1 o_2}_{\text{linear map } \mathcal{T}} \\ &+ \underbrace{o_1 \cdot \left(\frac{v_1(v_1+1)}{2} + v_1 o_1 \right) + o_2 \cdot \left(\frac{(v_1+o_1)\cdot(v_1+o_1+1)}{2} + (v_1+o_1)\cdot o_2 \right)}_{\text{central map } \mathcal{F}} \end{aligned}$$

field elements.

Furthermore, we introduce an intermediate map $\mathcal{Q} = \mathcal{F} \circ \mathcal{T}$. Note that the three maps $\mathcal{F}$, $\mathcal{Q}$ and $\mathcal{P}$ can be represented by matrices in two different ways



$v_1 + 1 \leq i \leq v_2$



$v_2 + 1 \leq i \leq n$

Figure 3: Matrices $F^{(i)}$ $(i = v_1 + 1, \ldots, n)$

1. as Macaulay matrices M_F, M_Q and $M_P \in \mathbb{F}^{m \times D}$, where $D = \frac{n \cdot (n+1)}{2}$ is the number of terms in the Rainbow public key.

2. as a set of m $n \times n$ matrices $F^{(i)}$ $(i = v_1 + 1, \ldots, n)$ (or $Q^{(i)}$, $P^{(i)}$ respectively). We write these matrices as upper triangular matrices and divide them into submatrices as shown in equation (5).

$$Q^{(i)} = \begin{pmatrix} Q^{(i,1)}_{v_1 \times v_1} & Q^{(i,2)}_{v_1 \times o_1} & Q^{(i,3)}_{v_1 \times o_2} \\ 0_{o_1 \times v_1} & Q^{(i,5)}_{o_1 \times o_1} & Q^{(i,6)}_{o_1 \times o_2} \\ 0_{o_2 \times v_1} & 0_{o_2 \times o_1} & Q^{(i,9)}_{o_2 \times o_2} \end{pmatrix} \tag{5}$$

   (same with $F^{(i)}, P^{(i)}$).

In our algorithm we use both representations.

4.2 Efficient Key Generation of standard Rainbow

First, we choose a 256-bit seed $\mathbf{s}_{\text{priv}}$ and use a PRNG to generate from $\mathbf{s}_{\text{priv}}$ the matrices $S' \in \mathbb{F}^{o_1 \times o_2}$, $T^{(1)} \in \mathbb{F}^{v_1 \times o_1}$, $T^{(2)} \in \mathbb{F}^{v_1 \times o_2}$, $T^{(3)} \in \mathbb{F}^{o_1 \times o_2}$ as well as the non zero coefficients of the central map. These coefficients are written into the matrices $F^{(i)}$ $(i = v_1 + 1, \ldots, n)$ as shown in Figure 3. All elements in the white parts of the matrices are zero.

Computing the matrices $Q^{(i)}$

For the polynomials of the first layer we get from $Q^{(i)} = T^T \cdot F^{(i)} \cdot T$

$$MP = \begin{array}{|c|}\hline MP_1 \\ \hline MP_2 \\ \hline \end{array} \qquad MQ = \begin{array}{|c|}\hline MQ_1 \\ \hline MQ_2 \\ \hline \end{array}$$

Figure 4: Matrices MQ and MP for the standard Rainbow signature scheme

$$\begin{aligned}
Q_1^{(i)} &= F_1^{(i)}, \\
Q_2^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_1 + F_2^{(i)}, \\
Q_3^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + F_2 \cdot T_3, \\
Q_5^{(i)} &= \mathrm{UT}(T_1^T \cdot F_1^{(i)} \cdot T_1 + T_1^T \cdot F_2^{(i)}), \\
Q_6^{(i)} &= T_1^T (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + T_1^T \cdot F_2^{(i)} \cdot T_3 + (F_2^{(i)})^T \cdot T_2, \\
Q_9^{(i)} &= \mathrm{UT}(T_2^T \cdot F_1^{(i)} \cdot T_2 + T_2^T \cdot F_2^{(i)} \cdot T_3). \qquad (6)
\end{aligned}$$

For the polynomials of the second layer we get from $Q^{(i)} = T^T \cdot F^{(i)} \cdot T$

$$\begin{aligned}
Q_1^{(i)} &= F_1^{(i)}, \\
Q_2^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_1 + F_2^{(i)}, \\
Q_3^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + F_2^{(i)} \cdot T_3 + F_3^{(i)}, \\
Q_5^{(i)} &= \mathrm{UT}(T_1^T \cdot F_1^{(i)} \cdot T_1 + T_1^T \cdot F_2^{(i)} + F_5^{(i)}), \\
Q_6^{(i)} &= T_1^T \cdot (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + T_1^T \cdot F_2^{(i)} \cdot T_3 \\
&+ T_1^T \cdot F_3^{(i)} + (F_2^{(i)})^T \cdot T_2 + (F_5^{(i)} + (F_5^{(i)})^T) \cdot T_3 + F_6^{(i)}, \qquad (7) \\
Q_9^{(i)} &= \mathrm{UT}(T_2^T \cdot F_1^{(i)} \cdot T_2 + T_2^T \cdot F_2^{(i)} \cdot T_3 + T_3^T \cdot F_5^{(i)} \cdot T_3 + T_2^T \cdot F_3^{(i)} + T_3^T \cdot F_6^{(i)}).
\end{aligned}$$

Here, $\mathrm{UT}(A)$ transforms the matrix A into an equivalent upper triangular matrix, that is, the matrix $A = (a_{ij})$ becomes $UT(A) = (\tilde{a}_{ij})$ with

$$\tilde{a}_{ij} = \begin{cases} a_{ij} + a_{ji} & \text{for } i < j \\ a_{ij} & \text{for } i = j \\ 0 & \text{otherwise.} \end{cases} \qquad (8)$$

Computing the matrix MQ

We define $m \times \frac{n \cdot (n+1)}{2}$ matrices MQ and MP of the form shown in Figure 4. We insert the elements of the matrices $Q^{(i)}$ $i = v_1 + 1, \ldots, n$ into the matrix MQ as follows

- The $\frac{n \cdot (n+1)}{2}$ non-zero elements of the matrix $Q^{(i)}$ are inserted into the $(i - v_1)$-th row of MQ as follows:

– The first $\frac{v_1 \cdot (v_1+1)}{2} + v_1 o_1$ positions of the $(i - v_1)$-th row are filled with the elements of the matrix $Q_1^{(i)} || Q_2^{(i)}$ (from left to right and top to bottom).
– The next $v_1 \cdot o_2$ positions are filled with the elements of $Q_3^{(i)}$ (again from left to right and top to bottom).
– The next $\frac{o_1 \cdot (o_1+1)}{2} + o_1 o_2$ positions are filled with the elements of $Q_5^{(i)} || Q_6^{(i)}$ (from left to right and top to bottom).
– The last $\frac{o_2 \cdot (o_2+1)}{2}$ positions of the row are filled with the elements of the matrix $Q_9^{(i)}$.

Computing the public key

Finally, we compute the matrix MP containing the coefficients of the public key by

$$\begin{aligned} MP_1 &= MQ_1 + S' \cdot MQ_2 \quad \text{and} \\ MP_2 &= MQ_2. \end{aligned}$$

Algorithm 9 shows the standard key generation process for the Rainbow signature scheme in compact form.

Algorithm 9 Efficient Key Generation of standard Rainbow

Input: linear transformations $\mathcal{S}, \mathcal{T}$ of form (3), matrices $F^{(i)}$ $(i = v_1+1, \ldots, n)$
Output: Rainbow public key $\mathcal{P}$ (consisting of the matrices MP_1 and MP_2)

1: **for** $i = v_1 + 1$ to v_2 **do**
2: Compute the matrices $Q_1^{(i)}, Q_2^{(i)}, Q_3^{(i)}, Q_5^{(i)}, Q_6^{(i)}, Q_9^{(i)}$ using equation (6).
3: **end for**
4: **for** $i = v_2 + 1$ to n **do**
5: Compute $Q_1^{(i)}$, $Q_2^{(i)}$, $Q_3^{(i)}$, $Q_5^{(i)}$, $Q_6^{(i)}$, $Q_9^{(i)}$ using equation (7).
6: **end for**
7: **for** $i = v_1 + 1$ to n **do**
8: Insert the elements of the matrix $Q^{(i)}$ into the $(i - v_1)$-th row of the matrix MQ (as described above)
9: **end for**
10: Compute the Rainbow public key by $MP_1 = MQ_1 + S' \cdot MQ_2$, $MP_2 = MQ_2$
11: **return** MP_1, MP_2.

4.3 Efficient Key Generation of CZ-Rainbow

For the key generation of CZ-Rainbow, we divide the matrices MQ and MP of Figure 4 into submatrices as shown in Figure 5. Here, $D_1 = \frac{v_1 \cdot (v_1+1)}{2} + v_1 o_1$ is the number of non-zero coefficients in the central polynomials of the first layer, $D_2 = \frac{v_2 \cdot (v_2+1)}{2}$ is the number of non-zero coefficients in the central polynomials

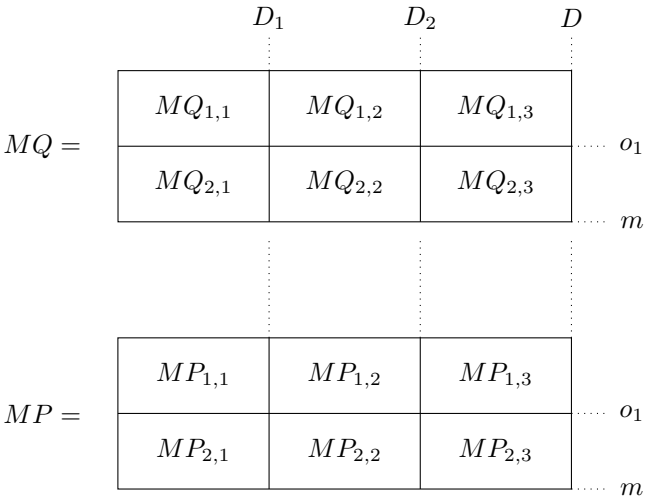


Figure 5: Matrices MQ and MP for CZ-Rainbow

of the second layer and $D = \frac{n\cdot(n+1)}{2}$ is the number of terms in the public polynomials.

We choose two 256 bit seeds $\mathbf{s}_{\text{priv}}$ and $\mathbf{s}_{\text{pub}}$. We use a PRNG to generate from $\mathbf{s}_{\text{priv}}$ the matrices S', $T^{(1)}$, $T^{(2)}$ and $T^{(3)}$ and from $\mathbf{s}_{\text{pub}}$ the matrices $MP_{1,1}$, $MP_{2,1}$ and $MP_{2,2}$ (see Figure 5).

First step: Compute the matrices $MQ_{(1,1)}$, $MQ_{(2,1)}$ and $MQ_{(2,2)}$

Due to the relation $\mathcal{P} = \mathcal{S} \cdot \mathcal{Q}$ we find

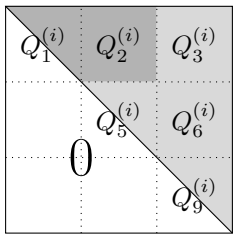
$$\begin{pmatrix} MQ_{1,1} \\ MQ_{2,1} \end{pmatrix} = S^{-1} \cdot \begin{pmatrix} MP_{1,1} \\ MP_{2,1} \end{pmatrix} = \begin{pmatrix} MP_{1,1} + S' \cdot MP_{2,1} \\ MP_{2,1} \end{pmatrix}$$

and

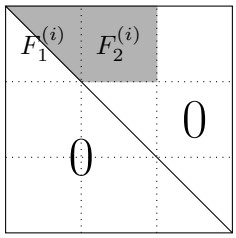
$$MQ_{2,2} = (S^{-1})_{22} \cdot B_2 = B_2.$$

Second Step: Compute the central polynomials of the first Rainbow layer

For this, we represent the first o_1 components of the map $\mathcal{Q}$ as upper triangular matrices $Q^{(i)}$ as shown in (5). Therefore, $Q^{(i)}$ looks as shown below.



We insert the D_1 elements of the i-th row of $MQ_{1,1}$ into the dark gray parts of the matrices $Q_1^{(i)}$ and $Q_2^{(i)}$ (from left to right and top to bottom). The light gray parts of the matrices $Q^{(i)}$ contain (yet unknown) elements of the field $\mathbb{F}$. The corresponding matrix $F^{(i)}$ representing the i-th central polynomial looks like



Here, the only non zero elements are located in the gray parts of $F_1^{(i)}$ and $F_2^{(i)}$. We consider the relation

$$F^{(i)} = (T^{-1})^T \cdot Q^{(i)} \cdot T^{-1}$$

and find that the elements of $F_1^{(i)}$ and $F_2^{(i)}$ only depend on the (already known) elements of $Q_1^{(i)}$ and $Q_2^{(i)}$. Such we get

$$\begin{aligned} F_1^{(i)} &= Q_1^{(i)}, \\ F_2^{(i)} &= (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_1 + Q_2^{(i)}. \end{aligned} \tag{9}$$

All the other elements of the matrices $F^{(i)}$ ($i \in \{1, \ldots, o_1\}$) are zero. So, after having determined the elements of $F_1^{(i)}$ and $F_2^{(i)}$, we can use the inverse relation

$$Q^{(i)} = T^T \cdot F^{(i)} \cdot T$$

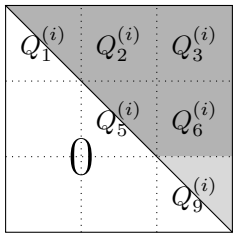
to compute the light gray parts of $Q^{(i)}$. We find

$$\begin{aligned} Q_3^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + F_2 \cdot T_3, \\ Q_5^{(i)} &= \mathrm{UT}(T_1^T \cdot F_1^{(i)} \cdot T_1 + T_1^T \cdot F_2^{(i)}), \\ Q_6^{(i)} &= T_1^T (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + T_1^T \cdot F_2^{(i)} \cdot T_3 + (F_2^{(i)})^T \cdot T_2, \\ Q_9^{(i)} &= \mathrm{UT}(T_2^T \cdot F_1^{(i)} \cdot T_2 + T_2^T \cdot F_2^{(i)} \cdot T_3). \end{aligned} \tag{10}$$

Here, $\mathrm{UT}(A)$ means transforming the square matrix A into an equivalent upper triangular matrix (c.f. Equation (8)).

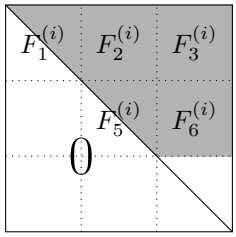
Third Step: Compute the central polynomials of the second Rainbow layer

For this, we represent the $o_1 + 1, \ldots, m$ components of the map $\mathcal{Q}$ as upper triangular matrices $Q^{(i)}$ as shown in (5). Therefore, $Q^{(i)}$ looks like



We insert the D_1 elements of the i-th row of $MQ^{(2,1)}$ into the dark gray parts of the matrices $Q_1^{(i)}$ and $Q_2^{(i)}$ (from left to right and top to bottom). The $D_2 - D_1$ elements of the i-th row of the matrix $MQ^{(2,2)}$ are inserted into the dark gray parts of the matrices $Q_3^{(i)}, Q_5^{(i)}$ and $Q_6^{(i)}$ (again left to right and top to bottom; i.e. we fill the matrix $Q_3^{(i)}$ first). The light gray part of the matrix $Q_9^{(i)}$ contains (yet unknown) elements of the field $\mathbb{F}$.

The corresponding matrix $F^{(i)}$ representing the i-th central polynomial looks like



Here, the only non zero elements are located in the gray parts of $F_1^{(i)}$, $F_2^{(i)}$, $F_3^{(i)}$, $F_5^{(i)}$ and $F_6^{(i)}$.

We consider the relation

$$F^{(i)} = (T^{-1})^T \cdot Q^{(i)} \cdot T^{-1}$$

and find that the elements of $F_1^{(i)}$, $F_2^{(i)}$, $F_3^{(i)}$, $F_5^{(i)}$ and $F_6^{(i)}$ only depend on the

already known elements of $Q^{(i)}$. Thus we get

$$\begin{aligned} F_1^{(i)} &= Q_1^{(i)}, \\ F_2^{(i)} &= (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_1 + Q_2^{(i)}, \\ F_3^{(i)} &= (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_4 + Q_2^{(i)} \cdot T_3 + Q_3^{(i)}, \\ F_5^{(i)} &= \mathrm{UT}(T_1^T \cdot Q_1^{(i)} \cdot T_1 + T_1^T \cdot Q_2^{(i)} + Q_5^{(i)}), \\ F_6^{(i)} &= T_1^T \cdot (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_4 + T_1^T \cdot Q_2^{(i)} \cdot T_3 \\ &+ T_1^T \cdot Q_3^{(i)} + (Q_2^{(i)})^T \cdot T_4 + (Q_5^{(i)} + (Q_5^{(i)})^T) \cdot T_3 + Q_6^{(i)} \end{aligned} \quad (11)$$

The other elements of the matrices $F^{(i)}$ ($i \in \{o_1 + 1, \ldots, m\}$) are zero. After having determined the elements of $F_1^{(i)}$, $F_2^{(i)}$, $F_3^{(i)}$, $F_5^{(i)}$ and $F_6^{(i)}$, we can use the inverse relation

$$Q^{(i)} = T^T \cdot F^{(i)} \cdot T$$

to compute the light gray parts of $Q^{(i)}$. We find

$$Q_9^{(i)} = \mathrm{UT}(T_2^T \cdot F_1^{(i)} \cdot T_2 + T_2^T \cdot F_2^{(i)} \cdot T_3 + T_3^T \cdot F_5^{(i)} \cdot T_3 + T_2^T \cdot F_3^{(i)} + T_3^T \cdot F_6^{(i)}). \quad (12)$$

Here, $\mathrm{UT}(A)$ means bringing the square matrix A into upper triangular form (c.f. Equation (8)).

Computing the remaining parts of the public key

For this last step, we transform the matrices $Q^{(i)}$ back into a Macaulay matrix M_Q. Here, the i-th row of the matrix M_Q contains all the elements of the matrix $Q^{(i)}$. In particular, the i-th rows of the matrices $MQ_{1,1}$ and $MQ_{2,1}$ contain the elements of the matrices $Q_1^{(i)}$ and $Q_2^{(i)}$ (read from left to right and from top to bottom). The i-th rows of the matrices $MQ_{1,2}$ and $MQ_{2,2}$ contain the elements of the matrices $Q_3^{(i)}$, $Q_5^{(i)}$ and $Q_6^{(i)}$ (again read from left to right and from top to bottom; i.e. the elements of $Q_3^{(i)}$ come first). The i-th row of the matrices $MQ_{1,3}$ and $MQ_{2,3}$ contains the elements of the matrix $Q_9^{(i)}$ (read from left to right and to bottom). Finally, we compute the matrix M_P by

$$M_P = S \cdot M_Q$$

or

$$\begin{aligned} MP_{1,2} &= MQ_{1,2} + S' \cdot MQ_{2,2}, \\ MP_{1,3} &= MQ_{1,3} + S' \cdot MQ_{2,3}, \\ MP_{2,3} &= MQ_{2,3} \end{aligned} \quad (13)$$

Algorithm 10 shows the key generation process of CZ-Rainbow in a compact form.

In CZ-Rainbow, the secret key and signing map has exactly the same form as for standard Rainbow. Also, in CZ-Rainbow, a majority of the verification (public map) time is taken up by the PRNG. Here, the default AES-based DRBG is not very fast, and a better DRBG would make the verification and key generation a lot faster (see also Subsection 6.5).

Algorithm 10 Efficient Key Generation of CZ-Rainbow

Input: linear transformations $\mathcal{S}, \mathcal{T}$ of form (3), matrices $B_1 \in \mathbb{F}^{m \times D_1}$ and $B_2 \in \mathbb{F}^{m \times (D_2 - D_1)}$

Output: Rainbow central map $\mathcal{S}$, matrices $MP_{1,2}$, $MP_{1,3}$, $MP_{2,3}$

$\begin{pmatrix} MQ_{1,1} \\ MQ_{2,1} \end{pmatrix} = S^{-1} \cdot B_1$ ▷ First Step

$MQ_{2,2} = B_2$

for $i = v_1 + 1$ to v_2 **do** ▷ Second Step

  Define an upper triangular matrix $Q^{(i)}$ of form (5).

  Insert the coefficients of the $(i - v_1)$-th row of the matrix $MQ_{1,1}$ into the submatrices $Q_1^{(i)}$ and $Q_2^{(i)}$.

  Set $F_1^{(i)} = Q_1^{(i)}$ and $F_2^{(i)} = (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_1 + Q_2^{(i)}$.

  Compute the matrices $Q_3^{(i)}, Q_5^{(i)}, Q_6^{(i)}, Q_9^{(i)}$ using equation (10).

end for

for $i = v_2 + 1$ to n **do** ▷ Third Step

  Define an upper triangular matrix $Q^{(i)}$ of form (5).

  Insert the coefficients of the $(i - v_1)$-th row of the matrix $MQ_{2,1}$ into the submatrices $Q_1^{(i)}$ and $Q_2^{(i)}$.

  Insert the coefficients of the $(i - v_1)$-th row of the matrix $MQ_{2,2}$ into the submatrices $Q_3^{(i)}, Q_5^{(i)}$ and $Q_6^{(i)}$.

  Compute $F_1^{(i)}$, $F_2^{(i)}$, $F_3^{(i)}$, $F_5^{(i)}$, $F_6^{(i)}$ using equation (11).

  Compute $Q_9^{(i)}$ using equation (12).

end for

for $i = v_1 + 1$ to n **do** ▷ Fourth Step

  Insert the elements of the matrix $Q^{(i)}$ into the $(i - v_1)$-th row of the matrix MQ (as described above)

end for

Compute the remaining parts of the public key by equation (13).

return $F^{(1)}, \ldots, F^{(m)}, MP_{1,2}, MP_{1,3}, MP_{2,3}$.

4.4 Key and Signature Generation of Compressed Rainbow

The key generation process of the compressed Rainbow scheme works in exactly the same way as that of the CZ-Rainbow scheme described above. However, we do not store the central map $\mathcal{F}$ computed by the algorithm, but only the seeds $\mathbf{s}_{\text{priv}}$ and $\mathbf{s}_{\text{pub}}$ used to generate the maps $\mathcal{S}$ an $\mathcal{T}$ as well as the matrices B_1 and B_2 used in the algorithm.

In order to sign a message / hash value, we first have to generate the full private key from these two seeds. We use $\mathbf{s}_{\text{priv}}$ to generate the matrices $\mathcal{S}$ and $\mathcal{T}$ of form (2) and $\mathbf{s}_{\text{pub}}$ to create the matrices B_1 and B_2 used in Algorithm 10. Next, we compute the central map $\mathcal{F}$ as shown in the algorithm (using line 1 to 6 as well as 9 to 13). Finally, we generate the signature in the same way as for the standard Rainbow scheme.

Since we have to perform parts of the key generation process during the signature generation, the signature generation of compressed Rainbow is much more inefficient than for standard and CZ-Rainbow.

5 Key Storage

In this section we describe the way how the public and private keys of Rainbow are stored in our implementations. We want to note that our recommendations here are not a must but only harmonize well with our implementations. When implementing Rainbow by yourself, feel free to develop your own format for the keys.

5.1 Representation of Finite Field Elements

5.1.1 GF(16)

Elements of GF(2) are stored as one bit 0 or 1. Elements of GF(4) are stored in two bits as linear polynomials over GF(2). The constant term of the polynomial is hereby stored in the least significant bit. Elements of GF(16) are stored in 4 bits as linear polynomials over GF(4). The constant term of the polynomial is hereby stored in the 2 least significant bits. Two adjacent GF(16) elements are packed into one byte. In a byte, the "lower" nibble is taken to come before the "higher" nibble.

5.1.2 GF(256)

Elements of GF(256) are stored in one byte as linear polynomials over GF(16). The constant term of the polynomial is hereby stored in the 4 least significant bits.

5.2 Public Key

The public key $\mathcal{P}$ of Rainbow is a system of m multivariate quadratic polynomials in n variables (we write $\mathcal{P} := \mathcal{MQ}(m, n)$).

We write it as a Macaulay matrix in column-major (the coefficients in different equations but of the same monomial are adjacent) form, with monomials ordered by lexicographic order $(x_1^2, x_1x_2, x_1x_3....x_1x_n, x_2^2 \ldots x_{n-1}^2, \ldots, x_{n-1}x_n, x_n^2)$.

$$\begin{aligned} y_1 &= q_{1,1,1}x_1x_1 + q_{1,2,1}x_1x_2 + \cdots + q_{1,n,1}x_1x_n + q_{2,2,1}x_2x_2 + \cdots \\ y_2 &= q_{1,1,2}x_1x_1 + q_{1,2,2}x_1x_2 + \cdots + q_{1,n,2}x_1x_n + q_{2,2,2}x_2x_2 + \cdots \\ \vdots &= \vdots \\ y_m &= q_{1,1,m}x_1x_1 + q_{1,2,m}x_1x_2 + \cdots + q_{1,n,m}x_1x_n + q_{2,2,m}x_2x_2 + \cdots \end{aligned} \tag{14}$$

Hereby, $q_{i,j,k}$ is the coefficient of the quadratic monomial x_ix_j of the polynomial y_k, where $i \leq j$. If we consider the indices of the coefficients $q_{i,j,k}$ as a 3-digit number, we order the coefficient with smaller indices in front. The coefficient sequence of the $\mathcal{MQ}(m, n)$ system (14) , is therefore stored (for the underlying fields GF(16) and GF(256)) in the form

$$[q_{1,1,1}, q_{1,1,2}, \ldots, q_{1,1,m}, q_{1,2,1}, \ldots, q_{1,n,m}, q_{2,2,1}, \ldots q_{n,n,m}].$$

CZ-Rainbow The public key of CZ (and Compressed) Rainbow contains first a 32-byte (256-bit) seed $\mathbf{s}_{pub}$. The AES counter mode DRBG of the reference implementation uses 48 bytes of AES key and nonce. Here, we use $\mathbf{s}_{pub}$ concatenated with the first 16 bytes of SHA256($\mathbf{s}_{pub}$) as the input to the AES-based DRBG. From this DRBG we read out $MP_{1,1}$, $MP_{2,1}$ and $MP_{2,2}$ in that order (cf. Section 4.3).

The remainder of the public key comprises $MP_{1,2}, MP_{1,3}, MP_{2,3}$ in that order. Within $MP_{1,2}$, we list the coefficients corresponding to those in matrices $Q_3^{(k)}, Q_5^{(k)}, Q_6^{(k)}$ in that order. $MP_{1,3}$ and $MP_{2,3}$ corresponds to coefficients in matrices $Q_9^{(k)}$. *Within each block, we order the coefficients as shown above.* That is, if we denote by $p_{i,j,k}$ the coefficient of x_ix_j (where $i \leq j$) in equation k, then within each block, the coefficients are sorted by k then i then j.

5.3 Private Key

The private key comprises the three components $\mathcal{S}, \mathcal{F}$, and $\mathcal{T}$. These components are stored in the order $\mathcal{S}, \mathcal{T}$, and $\mathcal{F}$.

5.3.1 The linear maps $\mathcal{S}$ and $\mathcal{T}$

The linear maps $\mathcal{S} : \mathbb{F}^m \to \mathbb{F}^m$ and $\mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$ are chosen to be of the form shown in equation (3) of Section 4. Moreover, we don't store the matrices $\mathcal{S}$ and

$\mathcal{T}$ itself, but their inverses $\mathcal{S}^{-1}$ and $\mathcal{T}^{-1}$. That is, we store (in this order) $S'_{o_1 \times o_2}$, $T^{(1)}_{v_1 \times o_1}, T^{(4)}_{v_1 \times o_2}$, and $T^{(3)}_{o_1 \times o_2}$. Within each block we store it in column-major form.

5.3.2 The central map $\mathcal{F}$

The central map $\mathcal{F}$ consists of two layers of quadratic equations. Recall that $\mathcal{F} = (f^{(v_1+1)}(\mathbf{x}), \ldots, f^{(n)}(\mathbf{x}))$ and

$$f^{(k)}(\mathbf{x}) = \sum_{i,j \in V_\ell, i \le j} \alpha^{(k)}_{ij} x_i x_j + \sum_{i \in V_\ell, j \in O_\ell} \beta^{(k)}_{ij} x_i x_j,$$

where $\ell \in \{1, 2\}$ is again the only integer such that $k \in O_\ell$.

For the first layer we have $V_1 := \{1, \ldots, v_1\}$ and $O_1 := \{v_1 + 1, \ldots, v_1 + o_1\}$, for the second layer $V_2 := \{1, \ldots, v_2 = v_1 + o_1\}$ and $O_2 := \{v_2 + 1, \ldots, n = v_2 + o_2\}$. The two layers of the central map $\mathcal{F}$ are stored separately.

While storing the first layer of $\mathcal{F}$, the coefficients of the equations $f^{(v_1+1)}, \ldots, f^{(v_2)}$ are further divided into parts denoted as F_1 ("vv") and F_2 ("vo") and are stored in the secret key in the order F_1 ("vv") followed by F_2 ("vo").

F_1 (vv) : The F_1 ("vv") part is an $\mathcal{MQ}(o_1, v_1)$ system, whose components are of the form

$$\sum_{i,j \in V_1, i \le j} \alpha^{(k)}_{ij} x_i x_j \quad \text{for } k \in O_1 \ .$$

It is stored in the same manner as the $\mathcal{MQ}(m, n)$ system of the public key (see Section 5.3). That is, $\alpha^{(k)}_{ij}$ is ordered by i first, then j, then by k.

Note that the F_1 ("vv") part contains the coefficients of the quadratic $v \times v$ terms.

The F_1 ("vv") part of the secret key of the first layer corresponds to the matrices $F_1^{(k)}$.

F_2 (vo) : The F_2 ("vo") part contains the remaining quadratic terms :

$$\sum_{i \in V_1} \sum_{j \in O_1} \beta^{(k)}_{ij} x_i x_j = \begin{bmatrix} x_{v_1+1}, \ldots, x_{v_1+o_1} \end{bmatrix} \begin{bmatrix} \beta^{(k)}_{11} & \ldots & \beta^{(k)}_{v_1 1} \\ \vdots & \ddots & \\ \beta^{(k)}_{1 o_1} & \ldots & \beta^{(k)}_{v_1 o_1} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{v_1} \end{bmatrix} \quad \text{for } k \in O_1 \ .$$

The F_2 ("vo") of the secret key of the first layer corresponds to the matrices $F_2^{(k)}$. The F_2 ("vo") part has its coefficients ordered just like F_1, so in the sequence

$$[\beta^{(v_1+1)}_{11}, \ldots, \beta^{(v_1+o_1)}_{11}, \beta^{(v_1+1)}_{12}, \ldots, \beta^{(v_1+o_1)}_{1n}, \beta^{(v_1+1)}_{22}, \ldots, \beta^{(v_1+o_1)}_{v_1 o_1}].$$

The coefficients of the second Rainbow layer are stored similarly. However, the parts are ordered corresponding to the matrices $F_1^{(k)}$, $F_2^{(k)}$, $F_3^{(k)}$, $F_5^{(k)}$, and

$F_6^{(k)}$, where $F_1^{(k)}$, $F_2^{(k)}$, $F_5^{(k)}$ comprise the "vv" part and $F_3^{(k)}$ and $F_6^{(k)}$ the "vo" part. Again, in each block we order the coefficients first by the corresponding monomials $x_i x_j$ (where $i \leq j$) and then by the equation index k.

CZ and Compressed Rainbow

In the case of CZ-Rainbow, the secret key is stored in exactly the same form as above. For Compressed Rainbow, the private key consists of the two 256 bit seeds $\mathbf{s}_{priv}$ and $\mathbf{s}_{pub}$ which are stored in this order. [3]

6 Implementation Details

In this Section we describe some details of our implementation. Besides speed, our main goal here is to achieve time-constancy for all operations depending on the private key.

6.1 Arithmetic over Finite Fields

6.1.1 The case of GF(16)

For multiplications over GF(16), our general strategy is the use of `VPSHUFB/TBL` for multiplication tables. While multiplying a bunch $\mathbf{a}$ of GF(16) elements stored in an SIMD register with a scalar $b \in$ GF(16), we load the table of results of multiplication with b and perform one `(V)PSHUFB` to get the result $\mathbf{a} \cdot b$.

Time-Constancy issues: Addressing table entries is a side-channel leakage. which reveals the value of b to a cache-time attack [7]. When time-constancy is needed, the straightforward method is again to use `VPSHUFB`. However, we do not use multiplication tables as above, but logarithm and exponentiation tables, and store the result in log-form if warranted. That is, we compute $a \cdot b = g^{(\log_g a + \log_g b)}$, and due to the characteristic of `(V)PSHUFB`, setting $\log_g 0 = -42$ is sufficient to make this operation time-constant even when multiplying three elements.[4] We shall see a different method below when working on a constant-time evaluation of a multivariate quadratic system over GF(16) (in the following sections, we denote this task shortly by "Evaluation of $\mathcal{MQ}$").

6.1.2 The case of GF(256)

Multiplications over GF(256) can be implemented using 2 table lookup instructions in the mainstream Intel SIMD instruction set. One `(V)PSHUFB` is used for the lower 4 bits, the other one for the top 4 bits.

[3] Note that we consider $\mathbf{s}_{pub}$ here as part of the private key, although it is publicly known. The reason for this is that $\mathbf{s}_{pub}$ is needed during the signature generation to recover the missing parts of the private key. Therefore, $\mathbf{s}_{pub}$ is considered as part of both the public and private key.

[4] Here, g is a generator of the multiplicative group GF(16).

Time-Constancy issues: For time-constant multiplications, we adopt the *tower field* representation of GF(256) which considers an element in GF(256) as a degree-1 polynomial over GF(16). The sequence of tower fields from which we build GF(256) is the following:

$$\begin{aligned} \mathrm{GF}(4) &:= \mathrm{GF}(2)[e_1]/(e_1^2 + e_1 + 1), \\ \mathrm{GF}(16) &:= \mathrm{GF}(4)[e_2]/(e_2^2 + e_2 + e_1), \\ \mathrm{GF}(256) &:= \mathrm{GF}(16)[e_3]/(e_3^2 + e_3 + e_2 e_1) \ . \end{aligned}$$

Using this representation, we can build constant-time multiplications over GF(256) from the techniques of GF(16). A time-constant GF(256) multiplication costs about 3 GF(16) multiplications for multiplying 2 degree-1 polynomials over GF(16) with the Karatsuba method and one extra table lookup instruction for reducing the degree-2 term.

6.2 The Public Map and Evaluation of $\mathcal{MQ}$

The public map of Rainbow is a straightforward evaluation of an $\mathcal{MQ}$ system. For pure public-key operations, the multiplications over GF(16) can be done by simply (1) loading the multiplication tables (`multab`) by the value of the multiplier and (2) performing a `VPSHUFB` for 32 results simultaneously. The multiplications over GF(256) can be performed with the same technique via 2 `VPSHUFB` instructions, using the fact that one lookup covers 4 bits.

Additionally, we make use of an entirely different public MQ evaluation technique due to Tung Chou: to compute the value of $\sum_i \mu_i \mathbf{v}_i$, where the μ_i are monomials, we keep a list of 15 accumulators $\mathbf{a}_g$, where $g \in \mathrm{GF}(16)$ and iterate over i, each time looking at μ_i, and do $\mathbf{a}_{\mu_i} \mathrel{+}= \mathbf{v}_i$. Only at the very end we compute $\sum g\mathbf{a}_g$.

6.2.1 Constant-Time Evaluation of $\mathcal{MQ}$ over GF(16) and GF(256)

While time-constancy issues are not important for the public key operations, we have to consider this issue during the evaluation of the "vv" terms of the central map $\mathcal{F}$.

In order to achieve time-constancy, we have to avoid loading `multab` according to a secret index for preventing cache-time attacks. To do this, we "generate" the desired `multab` instead of "loading" it by a secret value. More precisely, when evaluating $\mathcal{MQ}$ with a vector $\mathbf{w} = (w_1, w_2, \ldots, w_n) \in \mathrm{GF}(16)^n$, we can achieve a time-constant evaluation if we already have the `multab` of $\mathbf{w}$, which is $(w_1 \cdot \texttt{0x0}, \ldots, w_1 \cdot \texttt{0xf}), \ldots, (w_n \cdot \texttt{0x0}, \ldots, w_n \cdot \texttt{0xf})$, in the registers. [5]

In other words, instead of performing memory access indexed by a secret value, we perform a sequential memory access indexed by the index of variables to prevent revealing side-channel information.

[5] Given a natural basis $(b_0 = 1, b_1, \ldots)$ of a binary field GF(q), we represent b_j by 2^j for convenience. So b_1 is 2, $1 + b_1$ is 3, …, $1 + b_1 + b_2 + b_3$ is `0xF` for elements of GF(16), and analogously for larger fields; analogously, the AES field representation of GF(2^8) is called `0x11B` because it uses $x^8 + x^4 + x^3 + x + 1$ as irreducible polynomial.

We show the generation of `multab` for elements $\mathbf{w} \in \mathrm{GF}(16)$ in Figure 6. A further matrix-transposition-like operation is needed to generate the desired `multab`. The reason for this is that the initial byte from each register forms our first new table, corresponding to w_1, the second byte from each register is the table of multiplication by w_2, etc. Computing one of these tables costs 16 calls of `PSHUFB` and we can generate 16 or 32 tables simultaneously using the SIMD environment. The amortized cost for generating one `multab` is therefore one `PSHUFB` plus some data movements.

As a result, the constant-time evaluation of $\mathcal{MQ}$ over GF(16) or GF(256) is only slightly slower than the non-constant time version.

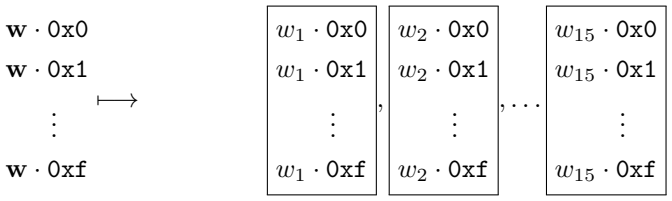


Figure 6: Generating `multab` for $\mathbf{w} = (w_1, w_2, \ldots w_{16})$. After $\mathbf{w} \cdot$ `0x0`, $\mathbf{w} \cdot$ `0x1`, ..., $\mathbf{w} \cdot$ `0xf` are calculated, each row stores the results of multiplications and the columns are the `multab` corresponding to $w_1, w_2, \ldots, w_{15}$. The `multab` of w_1, w_2, ...,w_{15} can be generated by collecting data in columns.

6.3 Gaussian Elimination in Constant Time

We use constant-time Gaussian elimination in the signing process of Rainbow. Constant-time Gaussian elimination was originally presented in [2] for GF(2) matrices and we extend the method to other finite fields. The problem of eliminations is that the pivot may be zero and one has to swap rows with zero pivots with other rows, which reveals side-channel information.

In order to test pivots against zero and switch rows in constant time, we can use the current pivot as a predicate for conditional moves and switch with every possible row, which can possibly contain non-zero leading terms. This constant-time Gaussian elimination is slower than a straightforward Gaussian elimination (see Table 1), but is still an $O(n^3)$ operation.

system	plain elimination	constant version
$\mathrm{GF}(16), 32 \times 32$	6,610	9,539
$\mathrm{GF}(256), 20 \times 20$	4,702	9,901

Table 1: Benchmarks on solving linear systems with Gauss elimination on Intel XEON E3-1245 v3 @ 3.40GHz, in CPU cycles.

7 Performance Analysis

7.1 Key and Signature Sizes

Tables 2 to 4 show the key and signature sizes for the proposed instances of the standard Rainbow scheme as well as CZ- and compressed Rainbow.

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	public key size (kB)	private key size (kB)	hash size (bit)	signature size (bit) [1]
I	(GF(16),36,32,32)	157.8	101.2	256	528
III	(GF(256),68,32,36)	861.4	611.3	576	1,312
V	(GF(256),96,36,64)	1,885.4	1,375.7	768	1,696

[1] 128 bit salt included

Table 2: Key and Signature Sizes for standard Rainbow

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	public key size (kB)	private key size (kB)	hash size (bit)	signature size (bit) [1]
I	(GF(16),36,32,32)	58.8	101.2	256	528
III	(GF(256),68,32,48)	258.4	611.3	576	1,312
V	(GF(256),96,36,64)	523.6	1,375.7	768	1,696

[1] 128 bit salt included

Table 3: Key and Signature Sizes for CZ-Rainbow

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	public key size (kB)	private key size (kB)	hash size (bit)	signature size (bit) [1]
I	(GF(16),36,32,32)	58.8	0.06	256	528
III	(GF(256),68,32,48)	258.4	0.06	576	1,312
V	(GF(256),96,36,64)	523.6	0.06	768	1,696

[1] 128 bit salt included

Table 4: Key and Signature Sizes for Compressed Rainbow

The following sections present the performance of our three Rainbow variants on the NIST reference platform (Section 7.2) as well as on two alternative platforms (Section 7.3).

7.2 Performance on the NIST Reference Platform

Tables 5 to 7 show the performance of our three Rainbow variants on the NIST reference platform. The data of this platform are given as follows.

Processor: Intel(R) Xeon(R) CPU E3-1275 v5 @ 3.60GHz (Skylake)
Clock Speed: 3.60GHz
Memory: 32GB (2x16) ECC DIMM DDR4 Synchronous 2133 MHz (0.5 ns)

Operating System: Linux 4.8.15, GCC compiler version 9.3
The NIST reference platform makes no use of special processor instructions.

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	32 M	319 k	41 k
	time (ms)	8.98	0.09	0.01
	memory(MB)	12.94	12.79	12.94
III	cycles	197 M	1.47 M	203 k
	time (ms)	54.59	0.41	0.06
	memory(MB)	15.09	13.40	13.64
V	cycles	436 M	2.48 M	362 k
	time (ms)	121.05	0.69	0.10
	memory(MB)	17.83	14.15	14.64

Table 5: Performance of standard Rainbow on the NIST Reference Platform (Linux/Skylake)

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	37 M	319 k	3.5 M
	time (ms)	10.26	0.09	0.98
	memory(MB)	12.79	12.79	12.79
III	cycles	232 M	1.47 M	20.4 M
	time (ms)	10.26	0.41	5.66
	memory(MB)	14.25	13.40	13.05
V	cycles	488 M	2.48 M	46 M
	time (ms)	135.62	0.69	12.81
	memory(MB)	16.00	14.15	13.31

Table 6: Performance of CZ-Rainbow on the NIST Reference Platform (Linux/Skylake)

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	37 M	19 M	3.5 M
	time (ms)	10.26	5.42	0.98
	memory(MB)	12.79	12.79	12.79
III	cycles	232 M	137 M	20.4 M
	time (ms)	10.26	5.42	5.66
	memory(MB)	14.25	13.40	13.05
V	cycles	488 M	233 M	46 M
	time (ms)	135.62	64.68	12.81
	memory(MB)	16.00	14.15	13.31

Table 7: Performance of Compressed Rainbow on the NIST Reference Platform (Linux/Skylake)

7.3 Performance on Other Platforms

In this section we present performance data of our three Rainbow variants on two alternative platforms using Skylake and Haswell processors respectively. Other than for the NIST Reference platform we make use of AVX2 vector instructions in this section.

7.3.1 Performance on the Skylake Platform

The data of the Skylake platform are given as follows.

Processor: Intel(R) Xeon(R) CPU E3-1275 v5 @ 3.60GHz (Skylake)
Clock Speed: 3.60GHz
Memory: 32GB (2x16) ECC DIMM DDR4 Synchronous 2133 MHz (0.5 ns)
Operating System: Linux 4.8.15, GCC compiler version 9.3
On the Skylake platform, we make use of AVX2 vector instructions.

Tables 8 to 10 show the performance of our three Rainbow variants on the Skylake platform.

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	9.9 M	67 k	34 k
	time (ms)	2.75	0.02	0.01
	memory(MB)	15.84	15.69	15.84
III	cycles	52 M	285 k	132 k
	time (ms)	14.59	0.08	0.04
	memory(MB)	18.16	16.30	16.55
V	cycles	192 M	739 k	392 k
	time (ms)	53.24	0.21	0.11
	memory(MB)	21.11	17.05	17.55

Table 8: Performance of standard Rainbow on Linux/Skylake (AVX2)

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	10.7 M	67 k	3.5 M
	time (ms)	2.98	0.02	0.97
	memory(MB)	15.69	15.69	15.69
III	cycles	64 M	285 k	20 M
	time (ms)	17.85	0.08	5.71
	memory(MB)	17.32	16.48	15.96
V	cycles	235 M	739 k	47 M
	time (ms)	65.23	0.21	13.09
	memory(MB)	19.31	17.43	16.22

Table 9: Performance of CZ-Rainbow on Linux/Skylake (AVX2)

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	10.7 M	7.0 M	3.5 M
	time (ms)	2.98	1.95	0.97
	memory(MB)	15.69	15.69	15.69
III	cycles	64 M	41 M	20 M
	time (ms)	17.85	11.41	5.71
	memory(MB)	17.32	16.48	15.96
V	cycles	235 M	118 M	47 M
	time (ms)	65.23	32.74	13.09
	memory(MB)	19.31	17.43	16.22

Table 10: Performance of Compressed Rainbow on Linux/Skylake (AVX2)

7.3.2 Performance on the Haswell Platform

The data of our Haswell platform are given as follows.

Processor: Intel(R) Xeon(R) CPU E7-8860 v3 @ 2.20GHz (Haswell)
Clock Speed: 2.2GHz
Memory: 2TB ECC DIMM DDR4 Synchronous 2133 MHz (0.5 ns)
Operating System: Linux 4.15.0-39, GCC compiler version 9.3
On the haswell platform, we make use of AVX2 vector instructions.

Tables 11 to 13 show the performance of our three Rainbow variants on the Haswell platform.

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	11 M	85 k	42 k
	time (ms)	5.07	0.04	0.02
	memory(MB)	12.99	12.84	12.99
III	cycles	60 M	348 k	162 k
	time (ms)	27.52	0.16	0.07
	memory(MB)	15.30	13.45	13.70
V	cycles	217 M	857 k	423 k
	time (ms)	98.87	0.39	0.19
	memory(MB)	18.25	14.20	14.70

Table 11: Performance of standard Rainbow on Linux/Haswell (AVX2)

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	12 M	85 k	3.5 M
	time (ms)	5.44	0.04	1.59
	memory(MB)	12.83	12.84	12.83
III	cycles	56 M	384 k	20 M
	time (ms)	25.59	0.16	9.20
	memory(MB)	14.46	13.45	13.11
V	cycles	207 M	857 k	45 M
	time (ms)	94.34	0.39	20.77
	memory(MB)	16.42	14.20	13.37

Table 12: Performance of CZ-Rainbow on Linux/Haswell (AVX2)

parameter set		key gen.	sign. gen.	sign. verif.
I	cycles	12 M	7 .6 M	3.5 M
	time (ms)	5.44	3.45	1.59
	memory(MB)	12.83	12.84	12.83
III	cycles	56 M	38 M	20 M
	time (ms)	25.59	17.26	9.20
	memory(MB)	14.46	13.62	13.11
V	cycles	207 M	108 M	45 M
	time (ms)	94.34	49.32	20.77
	memory(MB)	16.42	14.57	13.37

Table 13: Performance of Compressed Rainbow on Linux/Haswell (AVX2)

7.4 Note on the Measurements

Turboboost is disabled on our platforms. The main compilation flags are `gcc -O3 -std=c99 -Wall -Wextra (-mavx2)`. The used memory is measured during an actual run using `valgrind --tool=massif --pages-as-heap=yes` and include overheads such as system libraries. We take the average number of 500 runs for all measurement.

As expected, Skylake is superior to Haswell (which is almost the same as Broadwell) in standard Rainbow. It is to be noted that our CZ- and compressed Rainbow measurements are more about memory performance, since our Haswell platform is very good at it.

7.5 Performance on Embedded Platforms

In this section we present some performance data of our scheme on embedded platforms. We restrict ourselves here to Rainbow instance I.

Processor: ARM Cortex-M4 (Silicon Lab Giant Gecko EFM32GG11B)
Clock Speed: 16 MHz

parameter set		key gen. [cc]	sign. gen. [cc]	sign. verif. [cc]
I-Classic	w/o precomp.	151 590k	938k	238k
	w/ precomp.	151 590k	757k	238k
I-Circumzenithal	w/o precomp.	166 969k	940k	6 671k
	w/ precomp.	166 969k	753k	6 671k
I-Compressed	w/o precomp.	167 035k	77 803k	6 671k
	w/ precomp.	–	–	–

Table 14: Performance of Rainbow I on ARM Cortex-M4. Rainbow-Classic and Rainbow-Circumzenithal signing can be sped up by precomputing the bitsliced secret key.

7.6 Note on the Verification Timings

As can be seen from Tables 6, 9 and 12, the signature verification process of CZ- and compressed Rainbow is significantly slower than that of the standard Rainbow scheme. However, we want to note that this slowdown is caused completely by the use of the cryptographically secure, AES-based PRNG supplied by OpenSSL (which is the same as the NIST supplied one) to generate the "fixed" parts of the public key.

By using a faster stream cipher or even generating the public key using a Linear Feedback Shift Register (LFSR), this slowdown can be avoided nearly completely.

7.7 Trends as the number n of variables increases

Signing: The secret map involves Gaussian Elimination and time-constant MQ evaluation. Both are $O(n^3)$ operations.

Verification: The public map involves straightforward MQ evaluations, which are $O(n^3)$ operations (note: the public key size is also n^3).

Key Generation: Key generation is done via computing matrix products which is of order $O(n^2)$. The size of the resulting public key is $O(n^3)$.

These theoretical estimations match very well with the above experimental data (when looking at Rainbow instances over the same base field).

8 Expected Security Strength

The following table gives an overview over the 5 NIST security categories proposed in [21]. The three values for the number of quantum gates correspond to values of the parameter `MAXDEPTH` of 2^{40}, 2^{64} and 2^{96}.

category	$\log_2$ classical gates	$\log_2$ quantum gates
I	143	130 / 106 / 74
II	146	
III	207	193 / 169 / 137
IV	210	
V	272	258 / 234 / 202

Table 15: NIST security categories

All known attacks against Rainbow are basically classical attacks, some of which can be sped up by Grover's algorithm. Due to the long serial computation of Grover's algorithm and the large memory consumption of the attacks, we feel safe in choosing a value of `MAXDEPTH` between 2^{64} and 2^{96}.

8.1 General Remarks

The Rainbow signature scheme as described in Section 3.5 of this proposal fulfills the requirements of the EUF-CMA security model (existential unforgeability under chosen message attacks). The parameters of the scheme (in particular the length of the random salt) are chosen in a way that up to 2^{64} messages can be signed with one key pair. The scheme can generate signatures for messages of arbitrary length (as long as the underlying hash function can process them).

8.2 Practical Security

In this section we analyze the security offered by the parameter sets proposed in Section 3.8.

Since there is no proof for the practical security of Rainbow, we choose the parameters of the scheme in such a way that the complexities of the known attacks against the scheme (see Section 9) are beyond the required levels of security.

The formulas in Section 9 give complexity estimates for the attacks in terms of field multiplications. To translate these complexities into gate counts as proposed in the NIST specification, we assume the following.

- one field multiplication in the field GF(q) takes about $(\log_2 q)^2$ bit multiplications (AND gates) and the same number of additions (XOR gates).
- for each field multiplication performed in the process of the attack, we also need an addition of field elements. Each of these additions costs $\log_2 q$ bit additions (XOR).

Therefore, the number of gates required by an attack can be computed as

$$\#\text{gates} = \#\text{field multiplications} \cdot (2 \cdot (\log_2 q)^2 + \log_2 q).$$

The following tables show the security provided by the proposed Rainbow instances against

- direct attacks (Section 9.2)
- the MinRank attack (Section 9.3)
- the HighRank attack (Section 9.4)
- the UOV attack (Section 9.5) and
- the Rainbow Band Separation (RBS) attack (Section 9.6).

While the direct attack is a signature forgery attack, which has to be performed for each message separately, the MinRank, HighRank, UOV and RBS attack are key recovery attacks. After having recovered the Rainbow private key using one of these attacks, an adversary can generate signatures in the same way as a legitimate user.

For each parameter set and each attack, the first entry in the cell shows (the base 2 logarithm of) the number of classical gates, while the second entry (if available) shows (the base 2 logarithm of) the number of logical quantum gates needed to perform the attack. In each row, the value printed in bold shows the complexity of the best attack against the given Rainbow instance.

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	$\log_2$(#gates) direct	MinRank	HighRank	UOV	RBS
I	(GF(16),36,32,32)	164	162	150	157	**147**
		122	162	**86**	91	147

A collision attack against the hash function underlying the Rainbow instance I is at least as hard as a collision attack against SHA256 (see Section 3.6). Therefore, Rainbow instance I meets the requirements of security category I.

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	$\log_2$(#gates) direct	MinRank	HighRank	UOV	RBS
III	(GF(256),68,32,48)	234	228	410	437	**217**
		200	228	218	233	217

A collision attack against the hash functions underlying the Rainbow instances III is at least as hard as a collision attack against SHA384. Therefore, the Rainbow instance III meet the requirements of the security categories III and IV.

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	$\log_2$(#gates) direct	MinRank	HighRank	UOV	RBS
V	(GF(256),96,36,64)	**285**	296	539	567	296
		243	296	283	299	**281**

A collision attack against the hash functions underlying the Rainbow instance V is at least as hard as a collision attack against SHA512. Therefore, Rainbow instance V meets the requirements of the security category V.

8.2.1 Overview

The following table gives an overview of the security provided by our Rainbow instances. For each of the NIST security categories I, III, and V [21] it lists the proposed Rainbow instances meeting the corresponding requirements.

security category	GF(16)	GF(256)
I and II	I	-
III and IV	-	III
V	-	V

Table 16: Proposed Rainbow Instances and their Security Categories

8.3 Side Channel Resistance

In our implementation of the Rainbow signature scheme (see Section 6) all key dependent operations are performed in a time-constant manner. Therefore, our implementation is immune against timing attacks.

9 Analysis of Known Attacks

Known attacks against the Rainbow signature scheme include

- collision attacks against the hash function (Section 9.1)
- direct attacks (Section 9.2)
- the MinRank attack including the new approach of Bardet et al. (Section 9.3)
- the HighRank attack (Section 9.4)
- The Rainbow-Band-Separation (RBS) attack including the bipartite XL version (Section 9.6)
- The UOV attack (Section 9.5)

- The Subfield Differential (SDA) attack (Section 9.7)

In the presence of quantum computers, one also has to consider brute force attacks accelerated by Grover's algorithm (see Section 9.8).

In Section 9.9, we furthermore look on the security of our Rainbow variants CZ- and compressed Rainbow.

While direct and brute force attacks are signature forgery attacks, which have to be performed for every message separately, rank attacks as well as the RBS and UOV attack are key recovery attacks. After having recovered the Rainbow private key using one of these attacks, the attacker can generate signatures in the same way as a legitimate user.

9.1 Collision attacks against the hash function

Since the Rainbow signature scheme follows the Hash then Sign approach, it can be attacked by finding collisions of the used hash function. We do not consider specific attacks against hash functions here, but consider the used hash function $\mathcal{H}$ as a perfect random function $\mathcal{H} : \{0,1\}^\star \to \mathbb{F}^m$.

Therefore, in order to prevent a (classical) collision attack against the hash function used in the Rainbow scheme, the number m of equations in the public system of Rainbow must be chosen such that

$$m \cdot \log_2 q \geq \text{seclev},$$

where q is the cardinality of the finite field and seclev is the required level of security. In other words, in order to prevent collision attacks against the used hash function, the number m of equations in the public key (and the central map) of Rainbow must be chosen to be at least

$$m \geq \frac{2 \cdot \text{seclev}}{\log_2 q}.$$

By this choice of m, we ensure that a (classical) collision attack against the hash function used by the Rainbow scheme requires at least 2^{seclev} evaluations of the hash function $\mathcal{H}$.

9.2 Direct Attacks

The most straightforward attack against multivariate schemes such as Rainbow is the direct algebraic attack, in which the public equation $\mathcal{P}(\mathbf{z}) = \mathbf{h}$ is considered as an instance of the MQ-Problem. Since the public system of Rainbow is an underdetermined system with $n \approx 1.5 \cdot m$, the most efficient way to solve this equation is to fix $n - m$ variables to create a determined system before applying an algorithm such as XL or a Gröbner Basis technique such as F_4 or F_5 [15]. It can be expected that the resulting determined system has exactly one solution. In some cases one obtains even better results when guessing additional variables before solving the system (hybrid approach) [3]. The complexity of solving such

a system of m quadratic equations in m variables using an XL Wiedemann approach can be estimated as

$$\text{Complexity}_{\text{direct; classical}} = \min_k \left(q^k \cdot 3 \cdot \binom{m-k+d_{\text{reg}}}{d_{\text{reg}}}^2 \cdot \binom{m-k}{2} \right)$$

field multiplications, where d_{reg} is the so called degree of regularity of the system. As it was shown by experiments, the public systems of Rainbow behave very similar to random systems. We therefore can estimate the degree of regularity as the smallest integer d for which the coefficient of t^d in

$$\frac{(1-t^2)^m}{(1-t)^{m-k}}$$

is non-positive.

In the presence of quantum computers, the additional guessing step of the hybrid approach might be sped up by Grover's algorithm. By doing so, we can estimate the complexity of a quantum direct attack by

$$\text{Complexity}_{\text{direct; quantum}} = \min_k \left(q^{k/2} \cdot 3 \cdot \binom{m-k+d_{\text{reg}}}{d_{\text{reg}}}^2 \cdot \binom{m-k}{2} \right)$$

field multiplications. Here, the value of d_{reg} can be estimated as above.

9.3 The MinRank Attack

In the **MinRank** attack the attacker tries to find a linear combination of the public polynomials of minimal rank. In the case of Rainbow, such a linear combination of rank v_2 corresponds to a linear combination of the central polynomials of the first layer. By finding o_1 of these low rank linear combinations, it is therefore possible to identify the central polynomials of the first layer and to recover an equivalent Rainbow private key.

MinRank Problem: Given the m $n \times n$ matrices $Q_{v_1+1}, \ldots, Q_n$ representing the homogeneous quadratic parts of the public Rainbow polynomials, find a linear combination $Q = \sum_{i=v_1+1}^{n} \lambda_i Q_i$ of rank $\leq r$.

There exist different methods to solve the MinRank problem, including

- the linear algebra approach: Choose randomly a vector $\mathbf{v} \in \mathbb{F}^n$ and check if this vector is contained in the kernel of the matrix Q.
- Kipnis-Shamir method: Since the kernel of the matrix Q is $n-r$ dimensional, there exists a $n \times (n-r)$ matrix Y such that $Q \times Y = 0_{n \times (n-r)}$ holds. By choosing the kernel vectors in systematic form, we can write the matrix Y in the form $Y = \binom{Y}{1_{n-r}}$, which reduces the number of variables in the system to $n + nr$. Solving the MinRank problem therefore corresponds to solving a system of $n \cdot (n-r)$ quadratic equations in $n+nr$ variables. Since the system is highly overdetermined, we can often solve it using the relinearization technique.

- Minors Modelling: Set the $r+1$ minors of the matrix Q to 0 and solve the resulting system of high degree polynomials. Since the system is highly overdetermined, we can solve it using the relinearization technique.

The currently most efficient method to solve the MinRank problem was proposed by Bardet et al. in [1]. In this modelling, one considers the decomposition of the low rank matrix Q into $Q = S \cdot C$, where S is an $n \times r$ and C is an $r \times n$ matrix representing the row space of the matrix Q. One defines matrices $C'_j = \binom{r_j}{C}$ and sets the $r+1$-Minors of these matrices C_j to zero. Since the resulting system has much more equations than variables, we can solve it by linearization using Wiedemanns algorithm.

In particular, the number of equations in the system is given by $m \cdot \binom{n}{r+1}$, where $\binom{n}{r+1}$ is the number of $r+1$ minors of the matrix C'_j. The number of variables in the system is $(o_2+1) \cdot \binom{n}{r}$. Therefore, if

$$(o_2+1) \cdot \binom{n}{r+1} \geq (o_2+1) \cdot \binom{n}{r} - 1 \tag{15}$$

holds, we can solve the system using the Wiedemann algorithm. The complexity of solving this system is therefore given as

$$Compl_{MinRank} = 3 \cdot \left(\left((o_2+1) \cdot \binom{n'}{r} \right)^2 \cdot (r+1) \cdot (o_2+1) \right). \tag{16}$$

Here, a careful anlysis shows that we don't have to consider all n rows of the matrices C'_j to be able to solve the system. The number n' in 16 denotes the smallest number for which inequality 15 is fulfilled. The impact of this attack on Rainbow and the proposed parameter sets was carefully analyzed in Sections 3 to 5 of [31]. In this paper we also analyzed a hybrid version of the attack, in which one guesses some of the variables λ_i used in the MinRank problem before applying the method of Bardet et al.

However, one of the statements of the document was incorrect. We stated

> "The new equations also contain equations that can not be derived trivially from the KS equations without getting through finding mutants."

We discovered an error in J. Ding's toy-example programming test. What we can prove now is that all new equations can be derived directly from the KS equations. We will soon put out a new paper in order to give the details.

This shows to us, that the new MinRank attack is nothing but a new efficient and very clever way of organizing the known equations. This is done through the formal minors instead of the new set of variables in the kernel (and/or row basis) of the desired MinRank matrix.

We believe this is the best in terms of improving the MinRank attack from the theoretical perspective since minors are the only algebraic structure we can explore to speed up the computations.

9.4 The HighRank attack

The goal of the **HighRank** attack [8] is to identify the (linear representation of the) variables appearing the lowest number of times in the central polynomials (these correspond to the Oil-variables of the last Rainbow layer, i.e. the variables x_i with $i \in O_u$).

The complexity of this attack can be estimated as

$$\text{Complexity}_{\text{HighRank; classical}} = q^{o_u} \cdot \frac{n^3}{6}.$$

In the presence of quantum computers, we can speed up the searching step using Grover's algorithm. Such we get

$$\text{Complexity}_{\text{HighRank; quantum}} = q^{o_u/2} \cdot \frac{n^3}{6}.$$

field multiplications.

9.5 UOV - Attacks

Since Rainbow can be viewed as an extension of the well known Oil and Vinegar signature scheme [18], it can be attacked using all known UOV attacks. In particular the Reconciliation attack [12] and the UOV "Oil Subspace" attack of Kipnis and Shamir [19], where the latter one is more serious.

One considers Rainbow as an UOV instance with $v = v_1 + o_1$ and $o = o_2$. The goal of this attack is to find the pre-image of the so called Oil subspace $\mathcal{O}$ under the affine transformation $\mathcal{T}$, where $\mathcal{O} = \{\mathbf{x} \in \mathbb{F}^n : x_1 = \cdots = x_v = 0\}$. Finding this space allows to separate the oil from the vinegar variables and recovering the private key.

The complexity of this attack can be estimated as

$$\text{Complexity}_{\text{UOV}-\text{Attack; classical}} = q^{n-2o_2-1} \cdot o_2^4$$

field multiplications. Using Grover's algorithm, this complexity might be reduced to

$$\text{Complexity}_{\text{UOV}-\text{Attack; quantum}} = q^{\frac{n-2o_2-1}{2}} \cdot o_2^4$$

field multiplications.

9.6 Rainbow-Band-Separation Attack

The Rainbow-Band-Separation attack [12] aims at finding linear transformations $\mathcal{S}$ and $\mathcal{T}$ transforming the public polynomials into polynomials of the Rainbow form (i.e. Oil $\times$ Oil terms must be zero). To do this, the attacker has to solve several nonlinear multivariate systems. The complexity of this step is determined by the complexity of solving the first (and largest) of these systems, which consists of $n + m - 1$ quadratic equations in n variables. However, the polynomials in this system are not random quadratic polynomials, but there

exist two groups of variables X and Y such that the polynomials are bilinear in X and Y. In [24, 20], it was observed that we can solve these systems faster than random systems.

In particular, we get two variable sets X and Y of size $|X| = n_x = v_1 + o_1$ and $|Y| = n_y = o_2$. We have $m_x = m$ polynomials which are quadratic in the variables from X and $m_y = n - 1$ equations bilinear in the variables from X and Y.

For a given bi-degree (α, β), a lower bound on the complexity of solving this system using a block Wiedemann approach is given by

$$Compl_{RBS;\alpha,\beta} = 3 \cdot \mathcal{M}_{\alpha,\beta}(t,s)^2 \cdot (n_x + 1) \cdot (n_y + 1),$$

the complexity of the whole attack can therefore be estimated as

$$Compl_{RBS} = \min_{\alpha,\beta} \; 3 \cdot \mathcal{M}_{\alpha,\beta}(t,s)^2 \cdot (n_x + 1) \cdot (n_y + 1),$$

where $\mathcal{M}_{\alpha,\beta}$ denotes the number of monomials of bi-degree less or equal to (α, β).
The impact of this bipartite version of the RBS attack on Rainbow and our proposed parameter sets was very carefully analyzed in [31].

We want to mention here that the speed up of the bipartite RBS attack is not dramatic and only costs us a few bit of security.

9.7 Subfield Differential and Nested Subfield Differential Attacks

In 2017, Ward Beullens *et al.* proposed the Lifted Unbalanced Oil and Vinegar signature scheme [5], which is a modification to the Unbalanced Oil and Vinegar scheme by Patarin. The core design of LUOV is as follows:

Let $\mathbb{F}_{2^r}$ be a degree r extension field of $\mathbb{F}_2$. Let o and v be two positive integers such that $o < v$ and $n = o + v$. The central map $\mathcal{F} : \mathbb{F}_{2^r}^n \to \mathbb{F}_{2^r}^o$ is a quadratic map whose components $f^{(1)}, \ldots, f^{(o)}$ are in the form:

$$f^{(k)}(\mathbf{x}) = \sum_{i=1}^{v} \sum_{j=i}^{n} \alpha_{i,j}^{(k)} x_i x_j + \sum_{i=1}^{n} \beta_i^{(k)} x_i + \gamma^{(k)},$$

where the coefficients $\alpha_{i,j}^{(k)'}s$, $\beta_i^{(k)'}s$ and $\gamma^{(k)'}s$ are chosen randomly from the base field $\mathbb{F}_2$ **only**. As in standard UOV, to hide the Oil and Vinegar structure of these polynomials, an invertible linear map $\mathcal{T} : \mathbb{F}_{2^r}^n \to \mathbb{F}_{2^r}^n$ is used to mix the variables.

Ding *et al.* proposed the Subfield Differential Attack [13] to break the LUOV signature scheme. The key idea of the attack is to transform the public key $\mathcal{P}$ into a map over a subfield which is more efficient to work over but still contains a signature for a given message. Namely, maps of the form $\overline{\mathcal{P}} : \mathbb{F}_{2^d}^n \to \mathbb{F}_{2^r}^o$ defined by

$$\overline{\mathcal{P}}(\bar{\mathbf{x}}) = \mathcal{P}(\mathbf{x}' + \bar{\mathbf{x}})$$

where $\mathbf{x}'$ is a random point $\mathbb{F}_{2^r}^n$. For any irreducible polynomial $g(t)$ of degree $r/d = s$,

$$\mathbb{F}_{2^d}[t]/(g(t)) \cong \mathbb{F}_{2^r},$$

and $\mathbb{F}_{2^d}$ is embedded as the set of constant polynomials

There is also a discussion in [13] on the inapplicability of the Subfield Differential Attack on Unbalanced Oil and Vinegar, which shows that such attacks can not be applied to Unbalanced Oil and Vinegar or Rainbow. Please see [13] for details.

After Ding *et al.* proposed the Subfield Differential Attack [13], the LUOV team made a change of parameters of LUOV for the second round of the NIST post quantum standardization competition [4]. However, Ding *et al.* proposed a modification to the Subfield Differential Attack called the Nested Subset Differential Attack which fully breaks half of the parameter sets put forward [10]. They also showed via experiments that this attack is not just a theoretical attack but can practically break the level I parameters in less than 210 minutes. The Nested Subset Differential attack is a large improvement of the Subfield Differential Attack which can be used in real world circumstances.

In [10], there is also a discussion about the inapplicability of the Nested Subset Differential Attack on Unbalanced Oil and Vinegar, which shows again that such attacks can not be applied to Unbalanced Oil Vinegar and Rainbow. Please see [10] for details.

9.8 Quantum Brute-Force-Attacks

In the presence of quantum computers, a brute force attack against the scheme can be sped up drastically using Grover's algorithm. In [30] it was shown that we can solve a system of $m-1$ binary quadratic equations in $n-1$ binary variables using $m+n+2$ qubits, and evaluating a circuit of $2^{n/2} \cdot (2m(n^2+2n)+1)$ quantum gates. Another variant is also given, which solves the system using less qubits, but with a larger circuit of about 2 times more quantum gates. For example, when $n = m$, a binary system of m equations in m variables can be solved using

$$2^{m/2} \cdot 2 \cdot m^3$$

bit operations. In general, we expect due to Grover's algorithm a quadratic speed up of a brute force attack. To reach a security level of seclev bits, we therefore need at least

$$m \geq \frac{2 \cdot \text{seclev}}{\log_2 q}$$

equations.

However, this condition is already needed to prevent collision attacks against the hash function. Therefore, we do not consider quantum brute force attacks in the parameter choice of our Rainbow instances.

9.9 Security of CZ-Rainbow

During the key generation process of a public key cryptosystem, we usually generate the private key first and then compute the public key out of he private key. In [27], the idea of cyclicRainbow is proposed, where we allow some cyclic structure inside the Rainbow system. However, if we observe carefully, what the idea in [27] really shows is that actually the process of generating public key from private key is a partially reversible process. Therefore, we can randomly select parts of the public key and parts of the private key, and can calculate from this the rest of the public and private key accordingly. Since this process is reversible anyway, one can see easily that this change of key producing process does not have any impact on the security. The CZ-Rainbow scheme is based precisely on this idea of reversing process, and it is the reason why we call it circumzenithal (CZ) Rainbow .

Since part of the public key can be randomly selected anyway, in the CZ-Rainbow scheme, we use a cryptographic PRNG to generate this fixed part of the public key, which is random with no structure that can be used by any attacks against the scheme besides the usual Rainbow attacks.

For the standard Rainbow scheme, we also use a PRNG to generate the Rainbow central map, therefore we obtain in both cases exactly the same number of $q^{|\mathbf{s}|}$ different random looking Rainbow key pairs, where $|\mathbf{s}|$ is the length of the seed used during key generation.

In [27], as explained above, it is shown that we have a one-to-one correspondence between the central map and the fixed parts of the public key. [6]

For standard Rainbow, we choose a small random seed $\mathbf{s}_{pr}$ and use a pseudo random number generator $PRNG_1$ to extend $\mathbf{s}_{pr}$ to a Rainbow central map $\mathcal{F}$. Using the key generation process of Rainbow, we compute from this map a (random looking!) public key $\mathcal{P}$. To summarize, we will obtain the Rainbow key pair $(\mathcal{F}, \mathcal{P})$.

For CZ-Rainbow, we choose a small random seed $\mathbf{s}_{pub}$ and use a pseudo random number generator $PRNG_2$ to extend $\mathbf{s}_{pub}$ to the matrices B_1 and B_2 of the CZ-Rainbow public key. Then we use the key generation process of CZ-Rainbow to compute the corresponding central map.

We can choose $PRNG_2$ in such a way that it maps $\mathbf{s}_{pub}$ exactly to the matrices B_1 and B_2 of the public key $\mathcal{P}$. Note that, under the assumption that $\mathcal{P}$ is indistinguishable from a random system, an adversary not aware of this construction principle views $PRNG_2$ still as a cryptographic PRNG. When we now apply the key generation process of CZ-Rainbow to B_1 and B_2, we will obtain the same Rainbow key pair $(\mathcal{F}, \mathcal{P})$ as above.

In other words, the two key generation processes of standard Rainbow and CZ-Rainbow are equivalent and, by choosing the PRNG used in the key generation process of CZ-Rainbow in a clever way, we could modify the scheme in such a way that the key generation processes of standard and CZ-Rainbow lead

[6] In the following we assume that both the key generation process of standard Rainbow and the key generation process of CZ-Rainbow use the same technique to derive the linear maps $\mathcal{S}$ and $\mathcal{T}$.

to the same $q^{|\mathbf{s}|}$ random looking Rainbow key pairs.

In this sense, we can view CZ-Rainbow just as a different kind of implementation of Rainbow, with a different kind of managing the trade-off between key size and computation cost.

10 Advantages and Limitations

The main advantages of the Rainbow signature scheme are

- **Efficiency**. The signature generation process of Rainbow consists of simple linear algebra operations such as matrix vector multiplication and solving linear systems over small finite fields. Therefore, the Rainbow scheme can be implemented very efficiently and is one of the fastest available signature schemes [14].

- **Short signatures**. The signatures produced by the Rainbow signature scheme are of size only a few hundred bits and therefore much shorter than those of RSA and other post-quantum signature schemes (see Section 7.1).

- **Modest computational requirements**. Since Rainbow only requires simple linear algebra operations over a small finite field, it can be efficiently implemented on low cost devices, without the need of a cryptographic coprocessor [9].

- **Security**. Though there does not exist a formal security proof which connects the security of Rainbow to a hard mathematical problem such as MQ, we are quite confident in the security of our scheme. Rainbow is based on the well known UOV signature scheme, against which, since its invention in 1999, no attack has been found. Rainbow itself was proposed in 2005, and the last attack requiring a serious parameter change was found in 2008 (ten years ago).

  Since then, despite rigorous cryptanalysis, no fundamental attack improvement against Rainbow has been developed. The recent new attack on Rainbow band separation attack can be more or less viewed as a more refined analysis of the previous attack, which affect the security in a rather minor way. We furthermore note here that, in contrast to some other post-quantum schemes, the theoretical complexities of the known attacks against Rainbow match the experimental data very well. Therefore, overall we are confident in the security of the Rainbow signature scheme.

- **Simplicity.** The design of the Rainbow schemes is extremely simple. Therefore, it requires only minimum knowledge in algebra to understand and implement the scheme. This simplicity also implies that there are not many structures of the scheme which could be utilized to attack the scheme. Therefore it is very unlikely that there are additional structures that can be used to attack the scheme which have not been discovered during more than 12 years of rigorous cryptanalysis.

On the other hand, the main disadvantage of Rainbow is the **large size of the public keys**. The private key size is not a problem in our new implementations. The public key sizes of Rainbow are, for security levels beyond 128 bit, in the range of 100 kB-1 MB and therefore much larger than those of classical schemes such as RSA and ECC and some other post-quantum schemes. However, due to increasing memory capabilities even of medium devices (e.g. smartphones), we do not think that this will be a major problem. Furthermore, the public key size can be reduced using the CZ-Rainbow scheme by up to 70%. Therefore, for memory constraint devices, CZ-Rainbow might be a good alternative for standard Rainbow.

In the following, we will discuss the practical applicability of Rainbow for different use cases.

1. Application to Transport Layer Security (TLS) Protocols

   Researchers did try to use Rainbow in TLS. We know that the OQS OpenSSL team actually has used Rainbow in OQS OpenSSL without any issue [34].

   In this paper, the authors also mention that Rainbow can be useful in TLS and other protocols because of its small signature size. The public key is big but the signature is small, so Rainbow can help more than Dilithium or Falcon in use-cases where the public key is not transferred. Such cases use Rainbow in the root CA certificate, or in SCT signatures in the endpoint cert or in OCSP staples. Rainbow would improve performance in these occasions compared to using Dilithium or Falcon.

   On the other hand, the authors of [25] state "More specifically, the TLS client application was not able to process the size of a Rainbow certificate yielding an excessive message size error". However, we would like to point out that this is due to their implementation which has a key size restriction. The TLS itself does not have such a restriction. Therefore, Rainbow is actually still a viable option in TLS.

2. Certificate Transparency [17]

   We would also like to point out that Rainbow is a very suitable solution for dealing with Certificate Transparency due to its short signature and fast computations.

3. Secure Boot and Verified Boot

   When we boot an operating system, we need to verify that the code that is used for booting is honest. This is done by verifying a signature on the operating system code, which already resides on the device at the boot-time. From what we know, FPGA companies like Xilinx care a lot about this scenario specifically. Rainbow's fast verification will make such Secure Boot processes fast.

   In the case of verified boot, it requires cryptographically verifying all executable code and data that is part of the Android version being booted

before it is used. This includes the kernel (loaded from the boot partition), the device tree (loaded from the boot partition), system partition, vendor partition, and so on. Rainbow is very suitable for applications which need fast verification like reboot.

4. Applications in the Internet of Things (IOT) for which the public key is not needed to be sent each time.

   An example for this is a small device with limited computing power to broadcast signed data that can be verified by servers who know the public key of the device.

In scenarios in which the public key is known beforehand such as Secure Boot and some IOT applications, we can speed up the signature verification process of Rainbow further using the following idea: For each message to be verified, the verifier chooses randomly $k < m$ of the m public key polynomials beforehand. Since the public key is already known, this can be done offline before even receiving a signed message. During the verification process itself, the verifier checks if the corresponding k polynomial equations are fulfilled. It is important that the k polynomials are chosen completely at random and that this choice is unknown to the signer.

In order to forge a (partial) signature fulfilling these k equations, an adversary has to 1) choose a set of $\ell \geq k$ public polynomials and hope that this set contains the k polynomials chosen by the verifier and 2) solve the instance of the MQ-Problem given by these ℓ polynomial equations. Therefore, the complexity of a signature forgery attack is given as

$$\mathrm{comp}_{\mathrm{sigforge}}(q, m, n, k) = \min_{k \leq \ell \leq m} \binom{m}{\ell} \cdot \mathrm{comp}_{\mathrm{direct}}(q, \ell, n).$$

When computing the complexity of the direct attack, we have to consider that ℓ is much smaller than n, say $n = \alpha\ell$ for $\alpha \geq 2$. In this case, we can use the technique proposed in [32] to solve the underdetermined system of ℓ equations in the same time as a determined system of $\ell - \lfloor \alpha \rfloor + 1$ equations.

A careful analysis shows that, for Rainbow instance I, we can choose $k = 32$. By using this idea it therefore should be possible to speed up the verification process of our scheme by a factor of nearly 2. Since, for secure boot, the public key is known beforehand, the selection part can be done offline and therefore does not slow down the verification process. We will soon publish a paper providing the details of this idea.

To the best of our knowledge, this technique only works for multivariate schemes and cannot be used for any other type of (post-quantum) signature schemes. We do not use this technique in our current implementation, because it can be only used in special applications.

References

[1] Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray A. Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, Javier A. Verbel: Algebraic attacks for solving the Rank Decoding and MinRank problems without Groebner basis. CoRR abs/2002.08322 (2020)

[2] D.J. Bernstein, T. Chou, P. Schwabe: McBits: Fast constant-time code based cryptography. CHES 2013, LNCS vol. 8086, pp. 250 - 272. Springer, 2013.

[3] L. Bettale, J.-C. Faugére, L. Perret: Hybrid approach for solving multivariate systems over finite fields. Journal of Mathematical Cryptology, 3, pp. 177- 197, 2009.

[4] W. Beullens, B. Preneel, A. Szepieniec, F. Vercauteren: LUOV signature scheme proposal for NIST PQC project (Round 2 version), 2019.

[5] W. Beullens, B. Preneel: Field lifting for smaller uov public keys. INDOCRYPT 2017, pp. 227–246. Springer, 2017.

[6] O. Billet, H. Gilbert. Cryptanalysis of Rainbow: SCN 2006, LNCS vol. 4116, pp. 336 - 347. Springer, 2006.

[7] J. Bonneau, I. Mironov: Cache-Collision Timing Attacks Against AES. CHES 2006, LNCS vol. 4249, pp. 201 - 215. Springer, 2006.

[8] D. Coppersmith, J. Stern, S. Vaudenay: Attacks on the birational signature scheme. CRYPTO 1994, LNCS vol. 773, pp. 435 - 443. Springer, 1994.

[9] P. Czypek, S. Heyse, E. Thomae: Efficient implementations of MQPC's on constrained devices. CHES 2012, LNCS vol. 7428, pp. 374-389. Springer, 2012.

[10] J. Ding, J. Deaton, F. Vishakha, B.-Y. Yang: The nested subset differential attack: A practical direct attack against luov which forges a signature within 210minutes. Iacr eprint 2020/967, 2020. https://eprint.iacr.org/2020/967.

[11] J. Ding, D. Schmidt: Rainbow, a new multivariable polynomial signature scheme. ACNS 2005, LNCS vol. 3531, pp. 164 - 175. Springer, 2005.

[12] J. Ding, B.-Y. Yang, C.-H. O. Chen, M.-S. Che, C.-M. Cheng: New differential-algebraic attacks and reparametrization of Rainbow. ACNS 2008, LNCS vol. 5037, pp. 242 - 257. Springer, 2008.

[13] J. Ding, Z. Zhang, J. Deaton, K. Schmidt, F. Visakha: New attacks on lifted unbalanced oil vinegar. The 2nd NIST PQC Standardization Conference, 2019.

[14] eBACS: ECRYPT Benchmarking of Cryptographic Systems. https://bench.cr.yp.to

[15] J.-C. Faugére: A new efficient algorithm for computing Gröbner Bases (F4). Journal of Pure and Applied Algebra, 139:61 - 88, 1999.

[16] M. R. Garay, D. S. Johnson: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.

[17] Internet Engineering Task Force: Certificate Transparency. Request for Comments 6962. Available at `https://tools.ietf.org/html/rfc6962`

[18] A. Kipnis, J. Patarin, L. Goubin: Unbalanced Oil and Vinegar schemes. EUROCRYPT 1999, LNCS vol. 1592, pp. 206 - 222. Springer, 1999.

[19] A. Kipnis, A. Shamir: Cryptanalysis of the Oil and Vinegar signature scheme. CRYPTO 1998, LNCS vol. 1462, pp. 257 - 266. Springer, 1998.

[20] S. Nakamura, Y. Ikematsu, Y. Wang, J. Ding, T. Takagi: New Complexity Estimation on the Rainbow Band-Separation Attack. IACR eprint 2020/703.

[21] NIST: Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. Available at `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`

[22] J. Patarin: Cryptanalysis of the Matsumoto and Imai public key scheme of Eurocrypt88. CRYPTO 1995, LNCS vol. 963, pp. 248-261. Springer 1995.

[23] J. Patarin: The oil and vinegar signature scheme. Dagstuhl Workshop on Cryptography , September 1996.

[24] R. A. Perlner, D. Smith-Tone: Rainbow Band Separation is Better than we Thought. IACR eprint 2020/702.

[25] D. Sikeridis, P. Kampanakis, M. Devetsikiotis: Post Quantum Authentication in TLS 1.3. IACR eprint 2020/071. Available at `https://eprint.iacr.org/2020/071.pdf`.

[26] A. Petzoldt and S. Bulygin. Linear Recurring Sequences for the UOV Key Generation Revisited. ICISC 2012, LNCS vol. 7839, pp. 441-455, Springer 2012.

[27] A. Petzoldt, S. Bulygin, J. Buchmann: CyclicRainbow - a Multivariate Signature Scheme with a Partially Cyclic Public Key. INDOCRYPT 2010, LNCS vol. 6498, pp. 33 - 48. Springer, 2010.

[28] A. Petzoldt: Efficient Key Generation for the Rainbow Signature Scheme. PQCrypto 2020.

[29] K. Sakumoto, T. Shirai, H. Hiwatari: On Provable Security of UOV and HFE Signature Schemes against Chosen-Message Attack. PQCrypto 2011, LNCS vol. 7071, pp 68 - 82. Springer, 2011.

[30] P. Schwabe, B. Westerbaan: Solving Binary *MQ* with Grover's Algorithm. SPACE 2016, LNCS vol. 10076, pp. 303 - 322. Springer 2016.

[31] The Rainbow Team: Modified Parameters of Rainbow in Response to a Refined Analysis of the Rainbow Band Separation Attack by the NIST Team and the Recent New Minrank attacks (June 2020). Available at http://precision.moscito.org/by-publ/recent/rainbow-pars.pdf

[32] E. Thomae C. Wolf: Solving Underdetermined Systems of Multivariate Quadratic Equations Revisited. PKC 2012, LNCS vol. 7293, pp. 156 - 171. Springer, 2012.

[33] C. Wolf and B. Preneel. Equivalent keys in hfe, c^*, and variations. In *Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2005.

[34] Open Quantum Safe: OQSOpen SSL Readme. Available at `https://github.com/open-quantum-safe/openssl/blob/OQS-OpenSSL_1_1_1-stable/README.md`