Name of Proposal:

Rainbow

Principal Submitter:

Jintai Ding

email: jintai.ding@gmail.com
phone: 513 556 - 4024
organization: University of Cincinnati
postal address: 4314 French Hall, OH 45221 Cincinnati, USA

Auxiliary Submitters: Ming-Shing Chen, 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

Type: Signature scheme

Family: Multivariate Cryptography, SingleField schemes

1 Algorithm Specification

In this section we present the Rainbow signature scheme as proposed in [7].

1.1 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

1.2 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 1.1), 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

$$\begin{aligned} f^{(k)}(x_1, \ldots, x_n) &= \sum_{i,j \in V_\ell, i \leq j} \alpha_{ij}^{(k)} x_i x_j \\ &+ \sum_{i \in V_\ell, j \in O_\ell} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in V_\ell \cup O_\ell} \gamma_i^{(k)} x_i + \delta^{(k)}, \quad (1) \end{aligned}$$

  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 $\mathbb{F}$ elements.

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

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

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.

1.3 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**

1.4 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. However, choosing $u = 2$ leads to a better scheme wth 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. For ease of implementation and performance issues, it is further common to choose the size of the two Rainbow layers (i.e. the values of o_1 and o_2) to be equal. In our parameter recommendations, we follow these two

guidelines [1].

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

Algorithm 2 `RainbowKeyGen`: Key Generation of Rainbow

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

$m \leftarrow o_1 + o_2$
$n \leftarrow m + v_1$
repeat
    $M_S \leftarrow \texttt{Matrix}(q, m, m)$
until IsInvertible(M_S) == **TRUE**
$c_S \leftarrow_R \mathbb{F}^m$
$\mathcal{S} \leftarrow \texttt{Aff}(M_S, c_S)$
$InvS \leftarrow M_S^{-1}$
repeat
    $M_T \leftarrow \texttt{Matrix}(q, n, n)$
until IsInvertible(M_T) == **TRUE**
$c_T \leftarrow_R \mathbb{F}^n$
$\mathcal{T} \leftarrow \texttt{Aff}(M_T, c_T)$
$InvT \leftarrow M_T^{-1}$
$\mathcal{F} \leftarrow \texttt{Rainbowmap}(q, v_1, o_1, o_2)$
$\mathcal{P} \leftarrow \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$
sk $\leftarrow (InvS, c_S, \mathcal{F}, InvT, c_T)$
pk $\leftarrow \mathcal{P}$
return (sk, pk)

The possible input values of Algorithm `RainbowKeyGen` are specified in Section 1.8. $\texttt{Matrix}(q, m, n)$ 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 (1)). 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

[1] In the parameter set IIIb we slightly differ from this rule by choosing $o_2 > o_1$. The reason for this is that, in this case, an unbalanced design of the Rainbow layers provides higher security against quantum attacks.

$\mathbb{F}_q$-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).

1.5 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 [15]. The main difference is the use of a random

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** $\mathbf{h}'==\mathbf{h}$ **then**
4: **return TRUE**
5: **else**
6: **return FALSE**
7: **end if**

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 perfom 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: $\quad y_1, \ldots, y_{v_1} \leftarrow_R \mathbb{F}$
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: $\quad (\hat{F}, c_F) \leftarrow \mathtt{Aff}^{-1}(\hat{f}^{(v_1+1)}, \ldots, \hat{f}^{(n)})$
5: **until** IsInvertible$(\hat{F})$ == **TRUE**
6: $InvF = \hat{F}^{-1}$
7: **repeat**
8: $\quad r \leftarrow \{0,1\}^\ell$
9: $\quad \mathbf{h} \leftarrow \mathcal{H}(\mathcal{H}(d)||r)$
10: $\quad \mathbf{x} \leftarrow InvS \cdot (\mathbf{h} - c_S)$
11: $\quad (y_{v_1+1}, \ldots, y_{v_2}) \leftarrow InvF \cdot ((x_{v_1+1}, \ldots, x_{v_2}) - c_F)$
12: $\quad \hat{f}^{(v_2+1)}, \ldots, \hat{f}^{(n)} \leftarrow \hat{f}^{(v_2+1)}(y_{v_1+1}, \ldots, y_{v_2}), \ldots, \hat{f}^{(n)}(y_{v_1+1}, \ldots, y_{v_2})$
13: $\quad t, (y_{v_2+1}, \ldots, y_n) \leftarrow \text{Gauss}(\hat{f}^{(v_2+1)} = x_{v_2+1}, \ldots, \hat{f}^{(n)} = x_n)$
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: $\quad$ **return TRUE**
5: **else**
6: $\quad$ **return FALSE**
7: **end if**

Similar to [15] 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 [13], we choose the length of the salt r to be 128 bit (independent of the security level).

1.6 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 instances aimed at NIST security categories I and II (see Section 1.8), we use SHA256 as the underlying hash function. In the Rainbow instances for the security categories III/IV and V/VI, we use SHA384 and SHA512 respectively.

In case a slightly longer (non standard) hash output is needed for our scheme (Rainbow schemes over GF(31) and 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).

1.7 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 that 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). In the case of Rainbow over GF(31), we use a sort of "rejection sampling". We first generate a random byte, corresponding to an integer $v \in \{0, \ldots, 255\}$. We reject v, if $v < 8$. Otherwise, the random GF(31) element v' is given by $v' = v \bmod 31 \in \{0, \ldots, 31\}$.

1.8 Parameter Choice

We propose the following nine parameter sets for Rainbow

Ia $(\mathbb{F}, v_1, o_1, o_2) = (\text{GF}(16), 32, 32, 32)$ (64 equations, 96 variables)

Ib $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(31), 36, 28, 28)$ (56 equations, 92 variables)

Ic $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(256), 40, 24, 24)$ (48 equations, 88 variables)

IIIb $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(31), 64, 32, 48)$ (80 equations, 144 variables)

IIIc $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(256), 68, 36, 36)$ (72 equations, 140 variables)

IVa $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(16), 56, 48, 48)$ (96 equations, 152 variables)

Vc $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(256), 92, 48, 48)$ (96 equations, 188 variables)

VIa $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(16), 76, 64, 64)$ (128 equations, 204 variables)

VIb $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(31), 84, 56, 56)$ (112 equations, 196 variables)

The proposed parameter sets are denoted as follows: The roman number indicates the NIST security category which the given Rainbow instance aims at (see Section 5.2). The letter indicates the finite field used in the scheme (a for GF(16), b for GF(31) and c for GF(256)). Note that, in some cases, a given parameter set fulfills the requirements of more than one of the security categories. For example, the Rainbow instance Ib (using GF(31) as underlying field) was designed to meet the criteria of NIST security categories I and II.

Additionally, we give here four further parameter sets providing less security and demonstrating certain issues how the parameter choice affects the security of the scheme.

0a $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(256), 40, 16, 16)$
For the parameter set 0a, the number of equations (given by $o_1 + o_2$) contained in the public key is not large enough to prevent direct attacks against the scheme. In fact, the complexity of this attack against the given Rainbow instance is only about 2^{109} classical gate equivalents.

0b $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(256), 24, 24, 24)$
For the parameter set 0b, the value of v_1 is too small to prevent the Rainbow-Band-Separation attack. In fact, the complexity of this attack against the given Rainbow instance is only about 2^{119} classical gate equivalents.

0c $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(256), 40, 36, 12)$
While, for the parameter set 0c, both the number of equations and variables are the same as for parameter set Ic, it provides significantly less security. The reason for this is the small value of o_2, which makes the scheme vulnerable to the HighRank attack. The security of this Rainbow instance is therefore only about 2^{120} classical gate equivalents.

0d $(\mathbb{F}, v_1, o_1, o_2) = (\mathrm{GF}(256), 40, 8, 40)$
While, for the parameter set 0d, both the number of equations and variables are the same as for parameter set Ic, it provides significantly less

security. The reason for this is the high value of o_2, which makes the scheme vulnerable to the UOV attack. The security of this Rainbow instance is therefore only about 2^{84} classical gate equivalents.

1.8.1 Data Conversion between bitstrings and GF(31) elements

When using Rainbow with GF(31) as the underlying finite field, we have to convert sequences of GF(31) elements into bitstrings and vice versa. During key generation, the public and private keys are generated as sequences of GF(31) elements and have to be converted into bitstrings for storage. In the signature generation process, we have to convert the private key and the hash value of the document (bitstrings) into sequences of GF(31) elements, generate the signature as a sequence of GF(31) elements and store it as a bitstring. During signature varification, we have to convert the public key, the hash value and the signature into sequences of GF(31) elements before running Algorithm 8. In order to perform this conversion, we proceed as follows.

To convert a sequence of GF(31) elements into a bitstring, we store 3 GF(31) elements in two bytes. In case the number of GF(31) elements is not divisible by three, we store each of the last k mod 3 GF(31) elements into 8 bits. Therefore, the length of the bitstring needed to store a key or signature of k GF(31) elements can be computed as

$$\text{length}_{\text{bitstring}} = (k \text{ div } 3) \cdot 16 + (k \text{ mod } 3) \cdot 8 \text{ bits.}$$

To compute the length of a hash value fitting into k GF(31) elements, we compute

$$\text{length}_{\text{hash value}} = (k \text{ div } 5) \cdot 24 + (k \text{ mod } 5) \cdot 4 \text{ bits.}$$

This uses the fact that 5 GF(31) elements can be effficiently used to store 3 bytes, while every additional GF(31) element can cover four bits.

2 Key Storage

2.1 Representation of Finte Field Elements

2.1.1 GF(31)

Elements of GF(31) are stored as integers in the range of 0 to 30. Any number out of this range, for example 31, is considered as a format error.
In order to reduce the key size, we apply the "packing operations" described in Section 1.8.1.

2.1.2 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.

2.1.3 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.

2.2 Public Key

The public key $\mathcal{P}$ of Rainbow is a system of m multivariate quadratic poynomials in n variables (we write $\mathcal{P} := \mathcal{MQ}(m, n)$). The monomials are ordered in the graded-reverse-lexicographic order.

$$\begin{aligned}
y_1 &= q_{1,1,1}x_1x_1 + q_{2,1,1}x_2x_1 + q_{2,2,1}x_2x_2 + q_{3,1,1}x_3x_1 + q_{3,2,1}x_3x_2 + \cdots \\
&+ l_{1,1}x_1 + l_{2,1}x_2 + \cdots + l_{n,1}x_n + c_1 \\
y_2 &= q_{1,1,2}x_1x_1 + q_{2,1,2}x_2x_1 + q_{2,2,2}x_2x_2 + q_{3,1,2}x_3x_1 + q_{3,2,2}x_3x_2 + \cdots \\
&+ l_{1,2}x_1 + l_{2,2}x_2 + \cdots + l_{n,2}x_n + c_2 \\
&\vdots \\
y_m &= q_{1,1,m}x_1x_1 + q_{2,1,m}x_2x_1 + q_{2,2,m}x_2x_2 + q_{3,1,m}x_3x_1 + q_{3,2,m}x_3x_2 + \cdots \\
&+ l_{1,m}x_1 + l_{2,m}x_2 + \cdots + l_{n,m}x_n + c_m
\end{aligned} \tag{2}$$

Hereby, $q_{i,j,k}$ is the coefficient of the quadratic monomial x_ix_j of the polynomial y_k, $l_{i,k}$ the coefficient of the linear monomial x_i in y_k and c_k the constant coefficient of y_k ($1 \leq j \leq i \leq n,\ 1 \leq k \leq m$).
In the key file we store the coefficients of the linear terms first, followed by those of the quadratic and constant terms.

If we consider the indices of the coefficients $l_{i,k}$, $q_{i,j,k}$ and c_k as 2-, 3- and 1-digit numbers respectively, we order the coefficient with smaller indices in front. The coefficient sequence of the $\mathcal{MQ}(m,n)$ system (2) , is therefore stored (for the underlying fields GF(16) and GF(256)) in the form

$$[l_{1,1}, l_{1,2}, \ldots, l_{1,m}, l_{2,1}, \ldots, l_{n,m}, q_{1,1,1}, q_{1,1,2}, \ldots, q_{1,1,m}, q_{2,1,1}, \ldots, q_{n,n,m}, c_1, \ldots, c_m].$$

2.2.1 The case of GF(31)

For the underlying field GF(31), we store the coefficients of the public key in a slightly different order to improve the performance of the verification process. While we still store the coefficients of the linear terms in front of those of the quadratic and constant terms, the coefficients of the linear and quadratic blocks are stored in "two-column" manner. By doing so, the coefficient sequence of the $\mathcal{MQ}(m,n)$ system (2) has the form

$$\begin{aligned}[\quad & l_{1,1}, l_{2,1}, l_{1,2}, l_{2,2}, \ldots, l_{1,m}, l_{2,m}, l_{3,1}, l_{4,1} \ldots, l_{n-1,m}, l_{n,m}, \\ & q_{1,1,1}, q_{2,1,1}, q_{1,1,2}, q_{2,1,2}, \ldots, q_{1,1,m}, q_{2,1,m}, q_{3,1,1}, q_{4,1,1}, \ldots, q_{n-1,n,m}, q_{n,n,m}, \\ & c_1, c_2, \ldots, c_m].\end{aligned}$$

The reason for storing the public key in this form is explained in Section 3.2.1. Note that, in order to be stored in this way, the number of variables in the public key must be even. However, for our parameter sets (see Section 1.8), this is the case.
After having generated the coefficient sequence, we apply the "packing operations" of Section 1.8.1 to reduce the size of the public key.

2.3 Secret Key

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

2.3.1 The affine maps $\mathcal{T}$ and $\mathcal{S}$

Suppose the affine map $\mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$ is given by

$$\mathcal{T}(\mathbf{x}) = \begin{bmatrix} t_{11} & t_{12} & \ldots & t_{1n} \\ & \vdots & \ddots & \\ t_{n1} & t_{n2} & \ldots & t_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} .$$

We store the matrix in column-major form. Hence, the affine map $\mathcal{T}$ is stored as a sequence

$$[t_{11}, t_{21}, \ldots, t_{n1}, t_{12}, \ldots, t_{nn}, c_1, , \ldots, c_n] .$$

The affine map $\mathcal{S} : \mathbb{F}^m \to \mathbb{F}^m$ is stored in the same manner.

2.3.2 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

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

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 3 parts denoted as "vv", "vo" , and "o-linear". They are stored in the secret key in the order "o-linear", followed by "vo" , and followed by "vv".

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

$$\sum_{i,j \in V_1, i \leq j} \alpha_{ij}^{(k)} x_i x_j + \sum_{i \in V_1} \gamma_i^{(k)} x_i + \delta^{(k)} \quad \text{for } k \in O_1 \ .$$

It is stored in the same manner as the $\mathcal{MQ}(m, n)$ system of the public key (see Section 2.3). Note that the "vv" part contains, additionally to the coefficients of the quadratic $v \times v$ terms, the coefficients of the terms linear in the Vinegar variables and all the constant terms of the map $\mathcal{F}$.

vo : The "vo" part contains the remaining quadratic terms which are

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

The "vo" part is stored in the form of o_1 column-major $o_1 \times v_1$ matrices, yielding the sequence

$$[\beta_{11}^{(v_1+1)}, \ldots, \beta_{1 o_1}^{(v_1+1)}, \beta_{21}^{(v_1+1)}, \ldots, \beta_{v_1 o_1}^{(v_1+1)}, \beta_{11}^{(v_1+2)}, \ldots, \beta_{v_1 o_1}^{(v_1+o_1)}].$$

o-linear : This part contains the coefficients of the remaining linear terms in the oil variables $x_{v_1+1}, \ldots, x_{v_2}$ of the first layer. These terms are given by

$$\begin{bmatrix} f_{v_1+1}(\mathbf{x}) \\ \vdots \\ f_{v_2}(\mathbf{x}) \end{bmatrix} = \ldots + \begin{bmatrix} \gamma_{v_1+1}^{(v_1+1)} & \ldots & \gamma_{v_2}^{(v_1+1)} \\ \vdots & \ddots & \\ \gamma_{v_1+1}^{(v_2)} & \ldots & \gamma_{v_2}^{(v_2)} \end{bmatrix} \begin{bmatrix} x_{v_1+1} \\ \vdots \\ x_{v_2} \end{bmatrix} \ .$$

The "o-linear" part is stored in the form of a row-major matrix, yielding the sequence

$$[\gamma_{v_1+1}^{(v_1+1)}, \ldots, \gamma_{v_2}^{(v_1+1)}, \gamma_{v_1+1}^{(v_1+2)}, \ldots, \gamma_{v_2}^{(v_2)}].$$

The coefficients of the second Rainbow layer are stored in the same way.

2.3.3 The case of GF(31)

For the underlying field GF(31), the coefficient sequence of the "vv" terms is stored in the same "two-column" format as the public key (see Section 2.2.1). Furthermore, all the column-major matrices in $\mathcal{T}$ and $\mathcal{S}$ as well as the "vo" part of $\mathcal{F}$ are also stored in "two-column" form. For example, the coefficients of the map $\mathcal{T}$ are stored as the sequence

$$[t_{11}, t_{1,2}, t_{2,1}, t_{2,2}, t_{3,1}, \ldots, t_{n,2}, t_{1,3}, \ldots, t_{n,n-1}, t_{n,n}, c_1, \ldots, c_n].$$

Note again that we use here the fact that n is an even number. The row-major matrix in "o-linear" is stored as described in the previous section.
After having generated the coefficient sequence of the private key, we apply the "packing operations" of Section 1.8.1 to reduce the key size.

3 Implementation Details

3.1 Arithmetic over Finite Fields

For GF(31), the straightforward use of arithmetic operations of the AVX2 instruction set is possible. We use `(V)PMULHRSW` (packed multiply high rounded signed word) to take remainders by 31 which is faster than using shifts.

3.1.1 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 follow with one `(V)PSHUFB` for 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 [4].
When time-constancy is needed, the straightforward method is again to use `VPSHUFB`. However, we do not use multpilication 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.[2] We shall see a different method below when working on an constant-time evaluation of a multivariate quadratic system over GF(16) (in the following sections, we denote this task shortly by "Evaluation of $\mathcal{MQ}$").

3.1.2 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.

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:

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

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) multplications 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.

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

The public map of Rainbow is a straightforward evaluation of an $\mathcal{MQ}$ system.

3.2.1 Evaluation of $\mathcal{MQ}$ over GF(31)

The matrix-like coefficients of $\mathcal{P}$ are stored as 8-bit values because we heavily rely on the AVX2 instruction `VPMADDUBSW`. In one instruction, `VPMADDUBSW` computes two 8-bit SIMD multiplications and a 16-bit SIMD addition.
All these operations are time-constant.
In order to perform these operations efficiently, we store the coefficients of the public key as described in Section 2.2.1.
Because `VPMADDUBSW` takes both a signed and an unsigned operand, one of the matrix and the monomial vector must be stored as signed bytes and one as unsigned bytes. Since $64 \cdot 31 \cdot 15 = 29760 < 2^{15}$, we can handle two `YMM` registers full of monomials before performing reductions on each individual accumulator. During computation, field elements are expressed as signed 16-bit values. If $m = 64$, we require 1024 bits of storage for each vector, precisely fitting in four 256-bit SIMD (`YMM`) registers.

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

To efficiently compute all polynomials for a given set of monomials, we keep all required data in registers and try to avoid register spilling throughout the computation, as much as possible.

3.2.2 Evaluation of $\mathcal{MQ}$ over GF(16) and GF(256)

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. Another trick is to multiply a vector of GF(16) elements by two GF(16) elements with one `VPSHUFB` since `VPSHUFB` can actually be seen as 2 independent `PSHUFB` instructions.

3.2.3 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. [3] 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.
We show the generation of `multab` for elements $\mathbf{w} \in \mathrm{GF}(16)$ in Figure 1. 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 1 `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.

[3] Note here and in the following. If we have 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, $\ldots$, $1 + b_1 + b_2 + b_3$ is `0xF` for elements of GF(16), and analogously for larger fields; using the same method, 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.

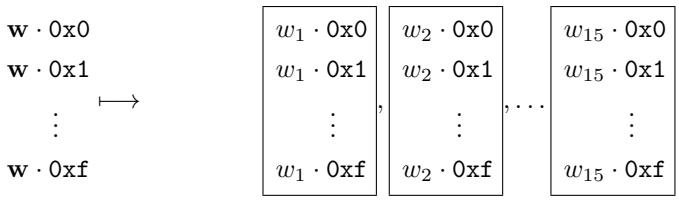

Figure 1: Generating `multab` for $\mathbf{w} = (w_1, w_2, \ldots w_{16})$. After $\mathbf{w} \cdot$ `0x0`, $\mathbf{w} \cdot$ `0x1`, ..., $\mathbf{w} \cdot$ `0xf` are calculated, each row stores the results of multiplications and the columns are the `multab` corresponding to $w_1, w_2, \ldots, w_{15}$. The `multab` of w_1, w_2, ...,w_{15} can be generated by collecting data in columns.

3.3 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 [1] 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. 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 straigtforward Gaussian elimination (see Table 1), but is still an $O(n^3)$ operation.

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

system	plain elimination	constant version
GF(16), 32×32	6,610	9,539
GF(31), 28×28	7,889	10,227
GF(256), 20×20	4,702	9,901

4 Performance Analysis

4.1 Key and Signature Sizes

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]
Ia	(GF(16),32,32,32)	148.5	97.9	256	512
Ib	(GF(31),36,28,28)	148.3	103.7	268	624
Ic	(GF(256),40,24,24)	187.7	140.0	384	832
IIIb	(GF(31),64,32,48)	512.1	371.4	384	896
IIIc	(GF(256),68,36,36)	703.9	525.2	576	1,248
IVa	(GF(16),56,48,48)	552.2	367.3	384	736
Vc	(GF(256), 92,48,48)	1,683.3	1,244.4	768	1,632
VIa	(GF(16), 76,64,64)	1,319.7	871.2	512	944
VIb	GF(31), 84,56,56)	1,321.0	922.4	536	1,176

[1] 128 bit salt included

Table 2: Key and Signature Sizes for Rainbow

4.2 Performance on the NIST Reference Platform

Processor: Intel(R) Xeon(R) CPU E3-1225 v5 @ 3.30GHz (Skylake)
Clock Speed: 3.30GHz
Memory: 64GB (4x16) ECC DIMM DDR4 Synchronous 2133 MHz (0.5 ns)
Operating System: Linux 4.8.5, GCC compiler version 6.4
No use of special processor instructions

parameter set		key gen.	sign. gen.	sign. verif.
Ia	cycles	1,302M	601k	350k
	time (ms)	394	0.182	0.106
	memory	3.3MB	3.0MB	2.6MB
Ib	cycles	4,578M	2,044k	1,944k
	time (ms)	1,387	0.619	0.589
	memory	3.6MB	3.3MB	2.9MB
Ic	cycles	4,089M	1,521k	939k
	time (ms)	1,239	0.461	0.285
	memory	3.3MB	3.0MB	2.8MB
IIIb	cycles	26,172M	5,471k	4,908k
	time (ms)	7,931	1.658	1.487
	memory	5.7MB	3.6MB	3.9MB
IIIc	cycles	31,612M	4,047k	2,974k
	time (ms)	9,579	1.226	0.901
	memory	4.6MB	2.9MB	3.1MB
IVa	cycles	11,176M	1,823k	1,241k
	time (ms)	3,387	0.552	0.376
	memory	4.3MB	3.0MB	2.8MB
Vc	cycles	116,046M	8,688k	6,174k
	time (ms)	35,165	2.633	1.871
	memory	7.0MB	3.7MB	3.9MB
VIa	cycles	45,064M	3,916k	2,897k
	time (ms)	13,655	1.187	0.878
	memory	6.1MB	3.8MB	3.8MB
VIb	cycles	164,689M	16,755k	11,224k
	time (ms)	49,906	5.077	3.401
	memory	10.3MB	4.4MB	6.0MB

Table 3: Performance of Rainbow on the NIST Refernce Platform (Linux/Skylake)

4.3 Performance on Other Platforms

Processor: Intel(R) Xeon(R) CPU E3-1225 v5 @ 3.30GHz (Skylake)
Clock Speed: 3.30GHz
Memory: 64GB (4x16) ECC DIMM DDR4 Synchronous 2133 MHz (0.5 ns)
Operating System: Linux 4.8.5, GCC compiler version 6.4
Use of AVX2 vector instructions

parameter set		key gen.	sign. gen.	sign. verif.
Ia	cycles	1,081M	75.5 k	25.5 k
	time (ms)	328	0.023	0.008
	memory	3.0MB	3.0MB	2.8MB
Ib	cycles	141M	426k	496k
	time (ms)	42.83	0.129	0.15
	memory	3.6MB	3.2MB	2.9MB
Ic	cycles	183M	111k	57.5k
	time (ms)	55.4	0.034	0.017
	memory	3.3MB	3.0MB	2.8MB
IIIb	cycles	813M	1,469k	1,791k
	time (ms)	246	0.445	0.543
	memory	5.9MB	4.1MB	4.1MB
IIIc	cycles	1,430M	326k	275k
	time (ms)	433	0.099	0.083
	memory	4.6MB	3.5MB	3.3MB
IVa	cycles	8,673M	899k	181k
	time (ms)	2,628	0.272	0.055
	memory	4.1MB	3.3MB	3.2MB
Vc	cycles	4,633M	616k	472k
	time (ms)	1,404	0.187	0.143
	memory	7.0MB	4.2MB	4.5MB
VIa	cycles	6,689M	575k	367k
	time (ms)	2,027	0.174	0.111
	memory	6.1MB	3.8MB	3.8MB
VIb	cycles	3,518M	3,655k	4690k
	time (ms)	1,066	1.108	1.421
	memory	10.0MB	5.3MB	6.0MB

Table 4: Performance of Rainbow on Linux/Skylake (AVX2)

Processor: Intel(R) Xeon(R) CPU D-1541 @ 2.10GHz (Broadwell)
Clock Speed: 2.1GHz
Memory: 64GB (4x16) ECC DIMM DDR4 Synchronous 2133 MHz (0.5 ns)
Operating System: Linux 4.8.5, GCC compiler version 6.4
Use of AVX2 vector instructions

parameter set		key gen.	sign. gen.	sign. verif.
Ia	cycles	1,147M	80.1k	27.0k
	time (ms)	546	0.038	0.013
	memory	3.0MB	2.9MB	2.7MB
Ib	cycles	150M	452k	526k
	time (ms)	71.4	0.216	0.250
	memory	3.7MB	3.2MB	2.8MB
Ic	cycles	194M	118k	61k
	time (ms)	92	0.056	0.029
	memory	3.1MB	3.2MB	2.7MB
IIIb	cycles	899M	1,282k	1,790k
	time (ms)	428	0.714	0.852
	memory	5.8MB	3.9MB	3.9MB
IIIc	cycles	1,535M	338k	274k
	time (ms)	736	0.161	0.130
	memory	4.6MB	3.4MB	3.3MB
IVa	cycles	9,161M	954k	212k
	time (ms)	3,464	0.454	0.101
	memory	4.2MB	3.2MB	3.1MB
Vc	cycles	5,019M	662k	472k
	time (ms)	2,390	0.315	0.225
	memory	6.9MB	4.1MB	4.2MB
VIa	cycles	45,182M	3,292k	2,899k
	time (ms)	13,882	0.998	0.878
	memory	6.7M	3.1M	3.5M
VIb	cycles	3,651M	3,697k	4,647k
	time (ms)	1,738	1.761	2.213
	memory	10.0MB	5.2MB	6.0MB

Table 5: Performance of Rainbow on Linux/Broadwell (AVX2)

4.4 Note on the measurements

Turboboost is disabled on our platforms. The main compilation flags are `gcc -O2 -std=c99 -Wall -Wextra (-mavx2)`. The used memory is measured during an actual run using `/usr/bin/time -f "%M"` (average of 10 runs). For key generation we take the average of 10 runs; for signature generation and verifi-

cation the average of 500 runs. As expected, Skylake is superior to Broadwell (which is almost the same as Haswell) in almost all cases.

4.5 Note on Rainbow schemes over GF(31)

For "b" (GF(31)) parameters, both the public map and the private map are on the order of a few cycles per byte of key, which represents an $\sim 1.5\times$ slowdown from GF(31) MQ evaluation in the literature. The main reason for this is that all keys are packed as mentioned in the implementation section. If we verify or (more likely) sign multiple times using the same key, this time goes down.

4.6 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 and in the case of "b" (GF(31)) parameters unpacking. These are $O(n^3)$ operations (note: the public key size is also n^3).

Key Generation: Key generation is done via the standard method (interpolation) which is of order $O(n^5)$, The size of the resulting public key is $O(n^3)$.

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

5 Expected Security Strength

The following table gives an overview over the 6 NIST security categories proposed in [13]. 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
VI	274	

Table 6: 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}.

5.1 General Remarks

The Rainbow signature scheme as described in Section 1.5 of this proposal fulfills the requirements of the EUF-CMA security model (existential unforeability 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).

5.2 Practical Security

In this section we analyze the security offered by the parameter sets proposed in Section 1.8.

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 6) are beyond the required levels of security.

The formulas in Section 6 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 6.2)
- the MinRank attack (Section 6.3)
- the HighRank attack (Section 6.4)
- the UOV attack (Section 6.5) and
- the Rainbow Band Separation (RBS) attack (Section 6.6).

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 $(\mathbb{F}, v_1, o_1, o_2)$	$\log_2(\#\text{gates})$				
		direct	MinRank	HighRank	UOV	RBS
Ia	(GF(16),32,32,32)	164.5	161.3	150.3	149.2	**145.0**
		146.5	95.3	**86.3**	87.2	145.0
Ib	(GF(31),36,28,28)	160.4	212.9	161.5	198.4	**148.1**
		129.3	121.2	**92.1**	111.7	145.6
Ic	(GF(256),40,24,24)	151.6	358.5	215.9	337.4	**148.2**
		130.8	194.5	**119.9**	181.4	148.2

A collision attack against the hash function underlying the Rainbow instances Ia, Ib and Ic is at least as hard as a collision attack against SHA256 (see Section 1.6). Therefore, the three Rainbow instances Ia, Ib and Ic meet the requirements of security category I. The Rainbow instances Ib and Ic also meet the requirements of security category II.

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	$\log_2$(#gates) direct	MinRank	HighRank	UOV	RBS
IIIb	(GF(31),64,32,48)	**214.4**	354.0	262.5	260.9	216.9
		179.4	193.0	**143.6**	144.5	214.5
IIIc	(GF(256),68,36,36)	215.2	585.1	313.9	563.8	**217.4**
		183.5	309.1	**169.9**	295.8	217.4

A collision attack against the hash functions underlying the Rainbow instances IIIb and IIIc is at least as hard as a collision attack against SHA384. Therefore, the Rainbow instances IIIb and IIIc 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
IVa	(GF(16),56,48,48)	233.4	259.9	216.3	247.5	**215.5**

A collision attack against the hash function underlying the Rainbow instance IVa is at least as hard as a collision attack against SHA384. Therefore, the Rainbow instance IVa meets the requirements of security category IV.

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	$\log_2$(#gates) direct	MinRank	HighRank	UOV	RBS
Vc	(GF(256),92,48,48)	**275.4**	778.8	411.2	747.4	278.6
		235.5	406.8	**219.2**	393.4	278.6

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

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	$\log_2$(#gates) direct	MinRank	HighRank	UOV	RBS
VIa	(GF(16),76,64,64)	302.6	341.6	281.6	329.2	**278.1**
VIb	(GF(31),84,56,56)	289.9	454.9	303.5	440.2	**279.8**

A collision attack against the hash functions underlying the Rainbow instances VIa and VIb is at least as hard as a collision attack against SHA512. Therefore, the Rainbow instances VIa and VIb meet the requirements of the security category VI.

5.2.1 Overview

The following table gives an overview of the security provided by our Rainbow instances. For each of the 6 NIST security categories [13] it lists the proposed

Rainbow instances meeting the corresponding requirements.

security category	GF(16)	GF(31)	GF(256)
I	Ia	Ib	Ic
II	-	Ib	Ic
III	-	IIIb	IIIc
IV	IVa	IIIb	IIIc
V	-	-	Vc
VI	VIa	VIb	VIc

Table 7: Proposed Rainbow Instances and their Security Categories

As can be seen from the table, we have, for each of the 6 NIST security categories, one Rainbow instance over the field GF(256). Over the smaller fields GF(16) and GF(31), some of the security categories (especially the quantum ones I, III and V) are not occupied. The reason for this are the large parameters needed to prevent in particular the quantum HighRank attack, which make these schemes inefficient.

5.3 Side Channel Resistance

In our implementation of the Rainbow signature scheme (see Section 3) all key dependent operations are performed in a time-constant manner. Therefore, our implementation is immune against timing attacks.

6 Analysis of Known Attacks

Known attacks against the Rainbow signature scheme include

- collision attacks against the hash function (Section 6.1)
- direct attacks (Section 6.2)
- the MinRank attack (Section 6.3)
- the HighRank attack (Section 6.4)
- The Rainbow-Band-Separation (RBS) atttack (Section 6.6)
- The UOV attack (Section 6.5)

In the presence of quantum computers, one also has to consider brute force attacks accelerated by Grover's algorithm (see Section 6.7).
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.

6.1 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 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 in the Rainbow scheme requires at least 2^{seclev} evaluations of the hash function $\mathcal{H}$.

6.2 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 [10]. 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) [2]. 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.

6.3 The MinRank Attack

In the **MinRank** attack [3] 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. As shown by Billet et al. [3], this step can be performed by

$$\text{Complexity}_{\text{MinRank; classical}} = q^{v_1+1} \cdot m \cdot \left(\frac{n^3}{3} - \frac{m^2}{6} \right) \tag{3}$$

field multiplications.

By the use of Grover's algorithm in the searching step, we can reduce this complexity to

$$\text{Complexity}_{\text{MinRank; quantum}} = q^{\frac{v_1+1}{2}} \cdot m \cdot \left(\frac{n^3}{3} - \frac{m^2}{6} \right) \tag{4}$$

field multiplications.

There exists an alternative formulation of the MinRank attack, the so called Minors Modelling. In this formulation, the MinRank problem is solved by solving a system of nonlinear polynomial equations (given by the $v_2 + 1$ minors of the matrix representing the required linear combination). The complexity of this attack can be estimated as

$$\text{Complexity}_{\text{MinRank; Minors}} = \binom{n + v_2 + 1}{v_2 + 1}^{\omega},$$

where $2 < \omega \leq 3$ is the linear algebra constant of solving a system of linear equations.
However, in the case of Rainbow, this complexity is higher than that of the MinRank attack using linear algebra techniques (see equation (3)). Furthermore, since we deal with a highly overdetermined system here, the MinRank attack using Minors Modelling can not be sped up by quantum techniques.

When analyzing the security of our Rainbow instances (see Section 5.2), we therefore use equations (3) and (4) to estimate the complexity of the MinRank attack.

6.4 The HighRank attack

The goal of the **HighRank** attack [5] 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.

6.5 UOV - Attack

Since Rainbow can be viewed as an extension of the well known Oil and Vinegar signature scheme [11], it can be attacked using the UOV attack of Kipnis and Shamir [12].
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 transfomation $\mathcal{T}$, 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.

6.6 Rainbow-Band-Separation Attack

The Rainbow-Band-Separation attack [8] 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. Since this is an overdetermined system (more equations than variables), we usually do not achieve a speed up by guessing variables before applying an algorithm like XL. However, in order to be complete, we consider the hybrid approach in our complexity estimate. Such we get

$$\text{Complexity}_{\text{RBS; classical}} = \min_k \cdot q^k \cdot 3 \cdot \binom{n + d_{\text{reg}} - k}{d_{\text{reg}}}^2 \cdot \binom{n-k}{2}$$

field multiplications. Again, the multivariate quadratic systems generated by this attack behave much like random systems. We can therefore estimate the value of d_{reg} as the smallest integer d, for which the coefficient of t^d in

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

is non-positive.

By using Grover's algorithm, we can speed up the guessing step of the hybrid approach. By doing so, we get

$$\text{Complexity}_{\text{RBS; quantum}} = \min_k \cdot q^{k/2} \cdot 3 \cdot \binom{n + d_{\text{reg}} - k}{d_{\text{reg}}}^2 \cdot \binom{n-k}{2}$$

field multiplications. The value of d_{reg} can be estimated as above.
However, as the optimal number k of variables to be guessed during the attack is very small (in most cases it is 0), the impact of quantum speed up on the complexity of the Rainbow-Band-Separation attack is quite limited.

6.7 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. For example, in [16] it was shown that 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.

7 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 [9].

- **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 4.1).

- **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 [6].

- **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 parameter change was found in 2008 (ten years ago). Since then, despite of rigorous cryptanalysis, no attack against Rainbow has been developed. We furthermore note here that, in contrast to some other post-quantum schemes, the theoretical complexities of the known attacks against Rainbow match very well the experimental data. So, all in all, we are quite 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 and private keys**. The (public and private) 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, we would like to point out that there exists a

technique to reduce the public key size of Rainbow by up to 65 % [14]. However such techniques in general come with the cost of a slower key generation process, and more important, these techniques often make the security analysis harder. This is why we do not want to apply these techniques for now. Nevertheless, in the future, we may apply these techniques, in particular, for special applications.

References

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

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

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

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

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

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

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

[8] 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.

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

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

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

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

[13] 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

[14] 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.

[15] 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.

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