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

The 2nd Round Proposal

Type: Signature scheme

Family: Multivariate Cryptography, SingleField schemes

1 Changes to the first round submission

We applied the following changes to our submission.

1. **Improvements on Key Generation Process**:

   We improved the key generation algorithm for the original Rainbow scheme as submitted in our first round proposal.

   In order to speed up the key generation process of Rainbow, we switched from computing the public key by interpolation to computing the public key using matrix products.
   Firstly, we restrict to homogeneous maps $\mathcal{S}$, $\mathcal{F}$ and $\mathcal{T}$. Note that this leads to a homogeneous public key $\mathcal{P}$. It is widely accepted that the complexity of direct attacks is determined only by the homogeneous part of highest degree. All other known attacks against Rainbow (see Section 8) explicitly use the (symmetric) matrices representing this homogeneous quadratic part. Therefore the security of Rainbow is not weakened by this modification.
   Secondly, we restrict the linear maps $\mathcal{S}$ and $\mathcal{T}$ to a special form. We use

   $$\mathcal{S} = \begin{pmatrix} 1_{o_1\times o_1} & S'_{o_1\times o_2} \\ 0_{o_2\times o_1} & 1_{o_2\times o_2} \end{pmatrix}$$

   and

   $$\mathcal{T} = \begin{pmatrix} 1_{v_1\times v_1} & T^{(1)}_{v_1\times o_1} & T^{(2)}_{v_1\times o_2} \\ 0_{o_1\times v_1} & 1_{o_1\times o_1} & T^{(3)}_{o_1\times o_2} \\ 0_{o_2\times v_1} & 0_{o_2\times o_1} & 1_{o_2\times o_2} \end{pmatrix}.$$

   Since for every Rainbow public key $\mathcal{P}$ there exists a corresponding Rainbow private key $(\mathcal{S},\mathcal{F},\mathcal{T})$ with $\mathcal{P} = \mathcal{S}\circ\mathcal{F}\circ\mathcal{T}$ and $\mathcal{S}$ and $\mathcal{T}$ being of the above form, the security of Rainbow is not weakened by this assumption. For a Rainbow scheme fulfilling these two conditions, we can perform the key generation process (both for the standard and modified variants which we will discuss below) using a number of matrix products (see Section 3) which enables us to speed up the Rainbow key generation process drastically.

2. **Parameter Choice**:
   Compared to the 9 recommended parameter sets of the first round submission, we narrow them to three parameter sets, namely

   - **Ia**: $\mathbb{F}$=GF(16), $(v_1, o_1, o_2) = (32, 32, 32)$ for the NIST security categories I and II,
   - **IIIc**: $\mathbb{F}$= GF(256), $(v_1, o_1, o_2) = (68, 36, 36)$ for the NIST security categories III and IV and
   - **Vc**: $\mathbb{F}$=GF(256), $(v_1, o_1, o_2) = (92, 48, 48)$ for the NIST security categories V and VI.

3. **Key Size and Performance Trade-off Variants**:

   We propose two variants of Rainbow, which make a trade-off in key size and performance. The first one of these is denoted as "cyclic Rainbow" and allows us to reduce the public key size of the scheme by up to 70 % at a higher cost of signature verification. The second version (denoted as "compressed Rainbow") furthermore stores the private key in the form of a 512 bit seed, thus enabling us to store the private key easily on small devices at the cost of the efficiency of the signature generation process. By proposing these Rainbow variants we want to illustrate the flexibility of the Rainbow signature scheme.

2 Algorithm Specification

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

2.1 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

2.2 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 2.1), 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

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

where $\ell \in \{1,\dots,u\}$ 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 $\mathbb{F}[x_1,\dots,x_n]$. The size of the public key is

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

field elements.

2.3 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 $\mathbf{y} = (y_1, \ldots, y_n)$
```

2.4 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 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. 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.
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 3 (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:     $M_S \leftarrow \texttt{Matrix}(q, m, m)$
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:     $M_T \leftarrow \texttt{Matrix}(q, n, n)$
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 2.8. `Matrix`(q, m, n) returns an $m \times n$ matrix with coefficients chosen uniformly at random in $\mathbb{F}_q$, while `Aff`(M, c) returns the affine map $M \cdot x + c$.
`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

$\mathbb{F}_q$-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: $\mathbf{y} \leftarrow$ InvF($\mathcal{F}, \mathbf{x}$)
4: $\mathbf{z} \leftarrow InvT \cdot (\mathbf{y} - c_T)$
5: return $\mathbf{z}$
```

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

```
Input: Rainbow central map $\mathcal{F} = (f^{(v_1+1)}, \dots, f^{(n)})$, 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:     $y_1, \dots, y_{v_1} \leftarrow_R \mathbb{F}$
3:     $\hat{f}^{(v_1+1)}, \dots, \hat{f}^{(n)} \leftarrow f^{(v_1+1)}(y_1, \dots, y_{v_1}), \dots, f^{(n)}(y_1, \dots, y_{v_1})$.
4:     $t, (y_{v_1+1}, \dots, y_{v_2}) \leftarrow$ Gauss($\hat{f}^{(v_1+1)} = x_{v_1+1}, \dots, \hat{f}^{(v_2)} = x_{v_2}$)
5:     if $t$ == TRUE then
6:         $\hat{f}^{(v_2+1)}, \dots, \hat{f}^{(n)} \leftarrow \hat{f}^{(v_2+1)}(y_{v_1+1}, \dots, y_{v_2}), \dots, \hat{f}^{(n)}(y_{v_1+1}, \dots, y_{v_2})$
7:         $t, (y_{v_2+1}, \dots, y_n) \leftarrow$ Gauss($\hat{f}^{(v_2+1)} = x_{v_2+1}, \dots, \hat{f}^{(n)} = x_n$)
8:     end if
9: until $t$ == TRUE
10: return $\mathbf{y} = (y_1, \dots, y_n)$
```

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).

2.5 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 [16]. 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 h'==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 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 $\ell$ 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 $\ell$ 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:     $(\hat{F}, c_F) \leftarrow \mathtt{Aff}^{-1}(\hat{f}^{(v_1+1)}, \ldots, \hat{f}^{(n)})$
 5: until IsInvertible($\hat{F}$) == TRUE
 6: $InvF = \hat{F}^{-1}$
 7: repeat
 8:     $r \leftarrow \{0,1\}^\ell$
 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:     $t, (y_{v_2+1}, \ldots, y_n) \leftarrow$ 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 $\sigma$
```

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 h' == h then
 4:     return TRUE
 5: else
 6:     return FALSE
 7: end if
```

Similar to [16] 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).

2.6 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 1a aimed at the NIST security categories I and II, we use SHA256 as the underlying hash function. For the Rainbow instances IIIc and Vc, aimed 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(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).

2.7 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).

2.8 Parameter Choice

We propose the following three parameter sets for Rainbow

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

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

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

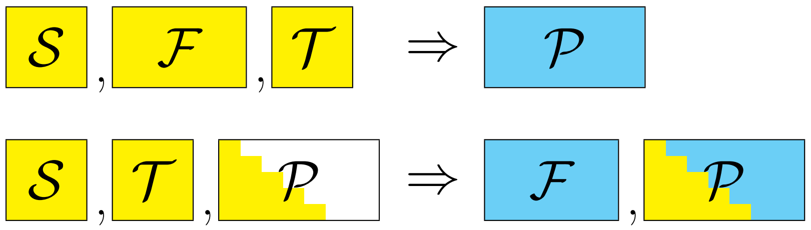

$\mathcal{S}$, $\mathcal{F}$, $\mathcal{T}$ $\Rightarrow$ $\mathcal{P}$
$\mathcal{S}$, $\mathcal{T}$, $\mathcal{P}$ $\Rightarrow$ $\mathcal{F}$, $\mathcal{P}$

Figure 1: Standard key generation (above) and alternative key generation for Rainbow. The yellow parts are chosen by the user, the blue parts are computed during the key generation process.

The proposed parameter sets are denoted as follows: The roman numeral indicates the NIST security category which the given Rainbow instance aims at (see Section 7.2). The letter indicates the finite field used in the scheme (a for GF(16) and c for GF(256))[1] Note that each of the given parameter sets fulfills the requirements of two of the NIST security categories. In particular, the Rainbow instance Ia meets the requirements of the security categories I and II, the parameter set IIIc is designed for security categories III and IV, and Vc meets the requirements of security categories V and VI.
For all three parameter sets we propose a standard as well as a cyclic and a compressed variant of the corresponding Rainbow instance.

2.9 Rainbow Variants

Additionally to the standard Rainbow scheme, we propose in this paper also a "cyclic" and a "compressed" variant of Rainbow. Here, the term "cyclic" does not mean that some parts of our keys can be represented by cyclic matrices. However, we use the term "cyclic" to indicate that our scheme is motivated by Petzoldt's cyclic Rainbow scheme of [15]. In [15], Petzoldt developed a technique to insert a structured matrix into the public key of the Rainbow scheme and to compute a corresponding private key (see Figure 1).
The technique not only allows to insert cyclic matrices into the public key, but also to create major parts of the public key from a small seed $\mathbf{s}_{\text{pub}}$ using a PRNG. In our proposal, we follow this approach. 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 [15] is far too inefficient for our purposes. Therefore, we developed a new key generation algorithm for the cyclic Rainbow scheme (see Section 3.3), which is only slighly

[1]The letter b originally referred to the underlying field GF(31). However, we omitted these scheme in the current proposal.

slower than the standard key generation process. However since, during the verification process, we have to decompress the public key before evaluating it, the verification process of the scheme is slowed down significantly. [2] Additionally to the standard and the "cyclic" variant, we also propose a Rainbow variant with a "compressed" key. This compressed Rainbow variant works very similar to the cyclic variant, but additionally stores the private key in the form of a 512 bit seed.

[2]However, we want to note that this slow down 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 slow down can be avoided nearly completely. Furthermore, additional structure in the public key might be used to speed up the evaluation of the public key further (see [14]). However, we believe that the security of Rainbow schemes with structured keys is not understood enough to propose them as a cryptographic standard. We therefore feel safer by using a public key which was generated by a cryptographically secure PRNG. However, we hope that proposing a non standard Rainbow variant here might motivate researchers to intensify their research on the security of Rainbow schemes with structured public key, too.

3 Efficient Key Generation

In this section we describe how to generate Rainbow key pairs in an efficient way. In the first subsection (Subsection 3.1) we introduce some general conventions and notations used in the remainder of this proposal. Subsection 3.2 then describes how to efficiently generate a key pair of the standard Rainbow signature scheme, while Subsection 3.3 deals with the key generation of the cyclicRainbow signature scheme. We note here that the term "cyclicRainbow" does not indicate that parts of the public key of our scheme are given by cyclic matrices. Instead we use this term to indicate that our scheme is motivated by the idea of cyclicRainbow [15]. In our proposed scheme, major parts of the public key can be generated from a small seed using a PRNG.

3.1 Conventions

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

- We restrict 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.
- We restrict 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

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

  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 (2) (c.f. [18]). So, the upper 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

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

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

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

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 (4).

$$Q^{(i)} = \begin{pmatrix} Q^{(i,1)}_{v_1 \times v_1} & Q^{(i,2)}_{v_1 \times o_1} & Q^{(i,3)}_{v_1 \times o_2} \\ 0_{o_1 \times v_1} & Q^{(i,5)}_{o_1 \times o_1} & Q^{(i,6)}_{o_1 \times o_2} \\ 0_{o_2 \times v_1} & 0_{o_2 \times o_1} & Q^{(i,9)}_{o_2 \times o_2} \end{pmatrix} \tag{4}$$

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

In our algorithm, we use both representations.

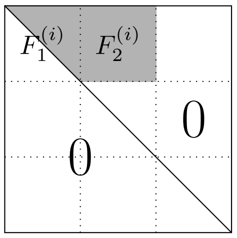
3.2 Efficient Key Generation of standard Rainbow

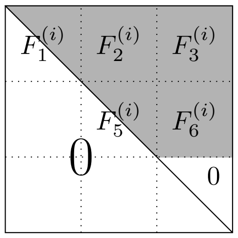
First, we choose a 256-bit seed $\mathbf{s}_{\text{priv}}$ and use a PRNG to generate from $\mathbf{s}_{\text{priv}}$ the matrices $S' \in \mathbb{F}^{o_1 \times o_2}$, $T^{(1)} \in \mathbb{F}^{v_1 \times o_1}$, $T^{(2)} \in \mathbb{F}^{v_1 \times o_2}$, $T^{(3)} \in \mathbb{F}^{o_1 \times o_2}$ as well 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 2.
All elements of the white parts of the matrices are zero.

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

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

$$\begin{aligned} Q_1^{(i)} &= F_1^{(i)}, \\ Q_2^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_1 + F_2^{(i)}, \\ Q_3^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + F_2 \cdot T_3, \\ Q_5^{(i)} &= \text{UT}(T_1^T \cdot F_1^{(i)} \cdot T_1 + T_1^T \cdot F_2^{(i)}, \\ Q_6^{(i)} &= T_1^T (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + T_1^T \cdot F_2^{(i)} \cdot T_3 + (F_2^{(i)})^T \cdot T_2, \\ Q_9^{(i)} &= \text{UT}(T_2^T \cdot F_1^{(i)} \cdot T_2 + T_2^T \cdot F_2^{(i)} \cdot T_3). \end{aligned} \tag{5}$$





$v_1 + 1 \leq i \leq v_2$ $\qquad$ $v_2 + 1 \leq i \leq n$

Figure 2: Matrices $F^{(i)}$ $(i = v_1 + 1, \ldots, n)$

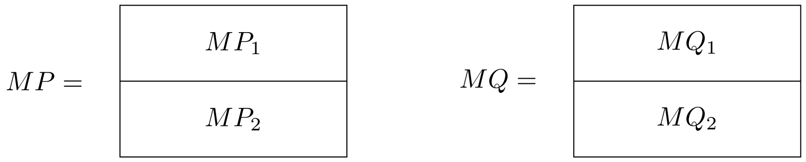


Figure 3: Matrices MQ and MP for the standard Rainbow signature scheme

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

$$\begin{aligned}
Q_1^{(i)} &= F_1^{(i)}, \\
Q_2^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_1 + F_2^{(i)}, \\
Q_3^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + F_2^{(i)} \cdot T_3 + F_3^{(i)}, \\
Q_5^{(i)} &= \mathrm{UT}(T_1^T \cdot F_1^{(i)} \cdot T_1 + T_1^T \cdot F_2^{(i)} + F_5^{(i)}, \\
Q_6^{(i)} &= T_1^T \cdot (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + T_1^T \cdot F_2^{(i)} \cdot T_3 \\
&+ T_1^T \cdot F_3^{(i)} + (F_2^{(i)})^T \cdot T_2 + (F_5^{(i)} + (F_5^{(i)})^T) \cdot T_3 + F_6^{(i)}, \qquad (6) \\
Q_9^{(i)} &= \mathrm{UT}(T_2^T \cdot F_1^{(i)} \cdot T_2 + T_2^T \cdot F_2^{(i)} \cdot T_3 + T_3^T \cdot F_5^{(i)} \cdot T_3 + T_2^T \cdot F_3^{(i)} + T_3^T \cdot F_6^{(i)}).
\end{aligned}$$

Here, $\mathrm{UT}(A)$ transforms the matrix A into an upper triangular matrix.

Computing the matrix MQ

We define $m \times \frac{n \cdot (n+1)}{2}$ matrices MQ and MP of the form shown in Figure 3. 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 (2), matrices $F^{(i)}$ $(i = v_1+1, \dots, n)$
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 (5).
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 $MP_1, MP_2$.
```

3.3 Efficient Key Generation of cyclicRainbow

For the key generation of cyclicRainbow, we divide the matrices MQ and MP of Figure 3 into submatrices as shown in Figure 4. 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

$$\begin{array}{r|c|c|c|l}
 & \quad\; D_1 & \quad\; D_2 & \quad\; D & \\
\hline
MQ = & MQ_{1,1} & MQ_{1,2} & MQ_{1,3} & \\
\cline{2-4}
 & MQ_{2,1} & MQ_{2,2} & MQ_{2,3} & o_1 \\
\cline{2-4}
 & & & & m
\end{array}$$

$$\begin{array}{r|c|c|c|l}
\hline
MP = & MP_{1,1} & MP_{1,2} & MP_{1,3} & \\
\cline{2-4}
 & MP_{2,1} & MP_{2,2} & MP_{2,3} & o_1 \\
\cline{2-4}
 & & & & m
\end{array}$$

Figure 4: Matrices MQ and MP for cyclic 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 4).

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

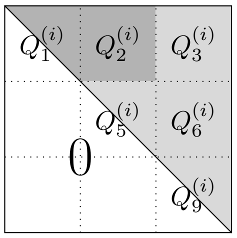
$$\begin{pmatrix} MQ_{1,1} \\ MQ_{2,1} \end{pmatrix} = S^{-1} \cdot \begin{pmatrix} MP_{1,1} \\ MP_{2,1} \end{pmatrix} = \begin{pmatrix} MP_{1,1} + S' \cdot MP_{2,1} \\ MP_{2,1} \end{pmatrix}$$

and

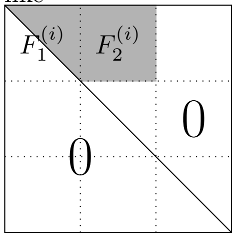
$$MQ_{2,2} = (S^{-1})_{22} \cdot B_2 = B_2.$$

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 (4). 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

$$\begin{aligned} F_1^{(i)} &= Q_1^{(i)}, \\ F_2^{(i)} &= (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_1 + Q_2^{(i)}. \end{aligned} \tag{7}$$

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

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

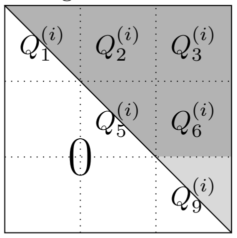
to compute the light gray parts of $Q^{(i)}$. We find

$$\begin{aligned} Q_3^{(i)} &= (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + F_2 \cdot T_3, \\ Q_5^{(i)} &= \mathrm{UT}(T_1^T \cdot F_1^{(i)} \cdot T_1 + T_1^T \cdot F_2^{(i)}, \\ Q_6^{(i)} &= T_1^T (F_1^{(i)} + (F_1^{(i)})^T) \cdot T_2 + T_1^T \cdot F_2^{(i)} \cdot T_3 + (F_2^{(i)})^T \cdot T_2, \\ Q_9^{(i)} &= \mathrm{UT}(T_2^T \cdot F_1^{(i)} \cdot T_2 + T_2^T \cdot F_2^{(i)} \cdot T_3). \end{aligned} \tag{8}$$

Here, $\mathrm{UT}(M)$ means bringing the square matrix M into upper triangular form.

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

For this, we represent the $o_1 + 1, \dots, m$ components of the map $\mathcal{Q}$ as upper triangular matrices $Q^{(i)}$ as shown in (4). Therefore, $Q^{(i)}$ looks as shown below.



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 $Q^{(i)}$. Such we get

$$\begin{aligned}
F_1^{(i)} &= Q_1^{(i)}, \\
F_2^{(i)} &= (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_1 + Q_2^{(i)}, \\
F_3^{(i)} &= (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_4 + Q_2^{(i)} \cdot T_3 + Q_3^{(i)}, \\
F_5^{(i)} &= \mathrm{UT}(T_1^T \cdot Q_1^{(i)} \cdot T_1 + T_1^T \cdot Q_2^{(i)} + Q_5^{(i)}, \\
F_6^{(i)} &= T_1^T \cdot (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_4 + T_1^T \cdot Q_2^{(i)} \cdot T_3 \\
&+ T_1^T \cdot Q_3^{(i)} + (Q_2^{(i)})^T \cdot T4 + (Q_5^{(i)} + (Q_5^{(i)})^T) \cdot T_3 + Q_6^{(i)} \qquad (9)
\end{aligned}$$

The other elements of the matrices $F^{(i)}$ ($i \in \{o_1 + 1, \dots, m\}$) are zero. So, 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

$$Q_9^{(i)} = \mathrm{UT}(T_2^T \cdot F_1^{(i)} \cdot T_2 + T_2^T \cdot F_2^{(i)} \cdot T_3 + T_3^T \cdot F_5^{(i)} \cdot T_3 + T_2^T \cdot F_3^{(i)} + T_3^T \cdot F_6^{(i)}). \quad (10)$$

Here, $\mathrm{UT}(M)$ means bringing the square matrix M into upper triangular form.

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 $i-th$ rows of the matrices $MQ_{1,3}$ and $MQ_{2,3}$ contain 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

$$\begin{aligned} MP_{1,2} &= MQ_{1,2} + S' \cdot MQ_{2,2}, \\ MP_{1,3} &= MQ_{1,3} + S' \cdot MQ_{2,3}, \\ MP_{2,3} &= MQ_{2,3} \qquad (11) \end{aligned}$$

Algorithm 10 shows the key generation process of cyclic Rainbow in a compact form.
In Cyclic Rainbow, the secret key and signing map has exactly the same form as for standard Rainbow.
Also, in Cyclic 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).

3.4 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 cyclic Rainbow scheme described above. However, we do not store the central map $\mathcal{F}$ computed by the algorithm, but only the seeds $\mathbf{s}_{\mathrm{priv}}$ and $\mathbf{s}_{\mathrm{pub}}$ 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 $\mathbf{s}_{\mathrm{priv}}$ to generate the matrices $\mathcal{S}$ and $\mathcal{T}$ of form (2) and $\mathbf{s}_{\mathrm{pub}}$ to create the matrices B_1 and B_2 used in Algorithm 10. Next, we compute the central map $\mathcal{F}$ as shown

Algorithm 10 Efficient Key Generation of cyclic Rainbow

Input: linear transformations $\mathcal{S}, \mathcal{T}$ of form (2), 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 $MP_{1,2}$, $MP_{1,3}$, $MP_{2,3}$

```
 1: $\begin{pmatrix} MQ_{1,1} \\ MQ_{2,1} \end{pmatrix} = S^{-1} \cdot B_1$        ▷ First Step
 2: $MQ_{2,2} = B_2$
 3: for $i = v_1 + 1$ to $v_2$ do        ▷ Second Step
 4:     Define an upper triangular matrix $Q^{(i)}$ of form (4).
 5:     Insert the coefficients of the $(i - v_1)$-th row of the matrix $MQ_{1,1}$ into the submatrices $Q_1^{(i)}$ and $Q_2^{(i)}$.
 6:     Set $F_1^{(i)} = Q_1^{(i)}$ and $F_2^{(i)} = (Q_1^{(i)} + (Q_1^{(i)})^T) \cdot T_1 + Q_2^{(i)}$.
 7:     Compute the matrices $Q_3^{(i)}, Q_5^{(i)}, Q_6^{(i)}, Q_9^{(i)}$ using equation (8).
 8: end for
 9: for $i = v_2 + 1$ to $n$ do        ▷ Third Step
10:     Define an upper triangular matrix $Q^{(i)}$ of form (4).
11:     Insert the coefficients of the $(i - v_1)$-th row of the matrix $MQ_{2,1}$ into the submatrices $Q_1^{(i)}$ and $Q_2^{(i)}$.
12:     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 $Q_6^{(i)}$.
13:     Compute $F_1^{(i)}$, $F_2^{(i)}$, $F_3^{(i)}$, $F_5^{(i)}$, $F_6^{(i)}$ using equation (9).
14:     Compute $Q_9^{(i)}$ using equation (10).
15: end for
16: for $i = v_1 + 1$ to $n$ do        ▷ Fourth Step
17:     Insert the elements of the matrix $Q^{(i)}$ into the $(i - v_1)$-th row of the matrix $MQ$ (as described above)
18: end for
19: Compute the remaining parts of the public key by equation (11).
20: return $F^{(1)}, \ldots, F^{(m)}, MP_{1,2}, MP_{1,3}, MP_{2,3}$.
```

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 cyclic Rainbow.

4 Key Storage

4.1 Representation of Finite Field Elements

4.1.1 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.

4.1.2 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.

4.2 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 $(x_1^2, x_1x_2, x_1x_3....x_1x_n, x_2^2 \ldots x_{n-1}^2, x_{n-1}x_n, x_n^2)$.

$$\begin{aligned}
y_1 &= q_{1,1,1}x_1x_1 + q_{1,2,1}x_1x_2 + \cdots + q_{1,n,1}x_1x_n + q_{2,2,1}x_2x_2 + \cdots \\
y_2 &= q_{1,1,2}x_1x_1 + q_{1,2,2}x_1x_2 + \cdots + q_{1,n,2}x_1x_n + q_{2,2,2}x_2x_2 + \cdots \\
\vdots &= \vdots \\
y_m &= q_{1,1,m}x_1x_1 + q_{1,2,m}x_1x_2 + \cdots + q_{1,n,m}x_1x_n + q_{2,2,m}x_2x_2 + \cdots
\end{aligned} \tag{12}$$

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 (12) , is therefore stored (for the underlying fields GF(16) and GF(256)) in the form

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

Cyclic version of Rainbow The public key of cyclic (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 3.3). 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 $Q_3^{(i)}, Q_5^{(i)}, Q_6^{(i)}$ in that order. $MP_{1,3}$ and $MP_{2,3}$ corresponds to coefficients in matrices $Q_9^{(i)}$. *Within each block, we order the coefficients as shown above.* That is, if we denote by $p_{i,j,k}$ the coefficient of $x_i x_j$ (where $i \leq j$) in equation k, then within each block, the coefficients are sorted by k then i then j.

4.3 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 $\mathcal{F}$.

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

The affine maps $\mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$ and $\mathcal{S} : \mathbb{F}^m \to \mathbb{F}^m$ are stored as in Section 3 (Equation 2). That is, we store $S'_{o_1 \times o_2}$, $T^{(1)}_{v_1 \times o_1}, T^{(4)}_{v_1 \times o_2}$, and $T^{(3)}_{o_1 \times o_2}$. Within each block, we store it column-major.

4.3.2 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

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

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 orderF_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

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

It is stored in the same manner as the $\mathcal{MQ}(m, n)$ system of the public key (see Section 4.3). That is, $\alpha_{ij}^{(k)}$ 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 :

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

The 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

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

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.

Cyclic and Compressed Rainbow

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

[3]Note that we consider $\mathbf{s}_{pub}$ here as part of the secret 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 secret key.

5 Implementation Details

5.1 Arithmetic over Finite Fields

5.1.1 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 $b \in \mathrm{GF}(16)$, 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.[4] 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 $\mathcal{M}Q$").

5.1.2 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:

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

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.

[4]Here, g is a generator of the multiplicative group $\mathrm{GF}(16)^\star$.

5.2 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. 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.

5.2.1 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.
We show the generation of `multab` for elements $\mathbf{w} \in \mathrm{GF}(16)$ in Figure 5. 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.

5.3 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)

[5] 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.

$$\begin{array}{c} \mathbf{w}\cdot\texttt{0x0}\\ \mathbf{w}\cdot\texttt{0x1}\\ \vdots\\ \mathbf{w}\cdot\texttt{0xf}\end{array} \longmapsto \boxed{\begin{array}{c} w_1\cdot\texttt{0x0}\\ w_1\cdot\texttt{0x1}\\ \vdots\\ w_1\cdot\texttt{0xf}\end{array}}, \boxed{\begin{array}{c} w_2\cdot\texttt{0x0}\\ w_2\cdot\texttt{0x1}\\ \vdots\\ w_2\cdot\texttt{0xf}\end{array}}, \dots \boxed{\begin{array}{c} w_{15}\cdot\texttt{0x0}\\ w_{15}\cdot\texttt{0x1}\\ \vdots\\ w_{15}\cdot\texttt{0xf}\end{array}}$$

Figure 5: Generating `multab` for $\mathbf{w} = (w_1, w_2, \dots w_{16})$. After $\mathbf{w}\cdot\texttt{0x0}$, $\mathbf{w}\cdot\texttt{0x1}$, $\dots$, $\mathbf{w}\cdot\texttt{0xf}$ are calculated, each row stores the results of multiplications and the columns are the `multab` corresponding to $w_1, w_2, \dots, w_{15}$. The `multab` of w_1, w_2, $\dots$,w_{15} can be generated by collecting data in columns.

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 straightforward 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
$\mathrm{GF}(16), 32\times 32$	6,610	9,539
$\mathrm{GF}(256), 20\times 20$	4,702	9,901

6 Performance Analysis

6.1 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)	149.0	93.0	256	512
IIIc	(GF(256),68,36,36)	710.6	511.4	576	1,248
Vc	(GF(256), 92,48,48)	1,705.5	1,227.1	768	1,632

[1] 128 bit salt included

Table 2: Key and Signature Sizes for Rainbow

parameter set	parameters $(\mathbb{F}, v_1, o_1, o_2)$	public key size (kB)	private key size (kB)[2]	hash size (bit)	signature size (bit) [1]
Ia	(GF(16),32,32,32)	58.1	93.0	256	512
IIIc	(GF(256),68,36,36)	206.7	511.4	576	1,248
Vc	(GF(256), 92,48,48)	491.9	1,227.1	768	1,632

[1] 128 bit salt included
[2] can be compressed to a seed of 512 bits (compressed Rainbow)

Table 3: Key and Signature Sizes for cyclic/ compressed Rainbow

6.2 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 4.8.15, GCC compiler version 6.4
No use of special processor instructions

parameter set		key gen.	sign. gen.	sign. verif.
Ia	cycles	35.0M	402K	155K
	time (ms)	10.6	0.122	0.0468
	memory	3.5MB	3.0MB	2.6MB
IIIc	cycles	340M	1.70M	1.64M
	time (ms)	103	0.516	0.497
	memory	4.6MB	2.9MB	3.1MB
Vc	cycles	757M	3.64M	2.39M
	time (ms)	229	1.10	0.723
	memory	7.0MB	3.7MB	3.9MB

Table 4: Performance of standard Rainbow on the NIST Reference Platform (Linux/Skylake)

parameter set		key gen.	sign. gen.*	sign. verif.
Ia	cycles	40.2M	20.2M	3.44M
	time (ms)	12.2	6.13	1.04
	memory	3.5MB	3.0MB	2.6MB
IIIc	cycles	402M	217M	19.4M
	time (ms)	122	65.8	5.89
	memory	4.6MB	2.9MB	3.1MB
Vc	cycles	879M	469M	45.4M
	time (ms)	266	142	13.7
	memory	7.0MB	3.7MB	3.9MB

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

* decompressing from 512-bit secret key (compressed Rainbow), otherwise the same as in Table 4

6.3 Performance on Other Platforms

Processor: Intel(R) Xeon(R) CPU E3-1275 v5 @ 3.60GHz (Skylake)
Clock Speed: 3.60GHz
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	8.29M	67.7K	21.7K
	time (ms)	2.30	0.019	0.006
	memory	3.5MB	3.0MB	2.8MB
IIIc	cycles	94.8M	588K	114K
	time (ms)	26.3	0.163	0.032
	memory	4.6MB	3.5MB	3.3MB
Vc	cycles	126M	755K	197K
	time (ms)	34.9	0.210	0.055
	memory	7.0MB	4.2MB	4.5MB

Table 6: Performance of standard Rainbow on Linux/Skylake (AVX2)

parameter set		key gen.	sign. gen.*	sign. verif.
Ia	cycles	9.28M	6.41M	3.37M
	time (ms)	2.58	1.781	0.936
	memory	3.5MB	3.0MB	2.8MB
IIIc	cycles	110M	61.8M	17.8M
	time (ms)	30.5	17.2	4.94
	memory	4.6MB	3.5MB	3.3MB
Vc	cycles	137M	87.2M	43.0M
	time (ms)	38.0	24.2	11.9
	memory	7.0MB	4.2MB	4.5MB

Table 7: Performance of cyclic/compressed Rainbow on Linux/Skylake (AVX2)

* decompressing from 512-bit secret key (compressed Rainbow), otherwise the same as in Table 6

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 7.3
Use of AVX2 vector instructions

parameter set		key gen.	sign. gen.	sign. verif.
Ia	cycles	8.90M	75.4K	26.9K
	time (ms)	4.05	0.034	0.012
	memory	3.5MB	2.9MB	2.7MB
IIIc	cycles	88.3M	654K	141K
	time (ms)	40.1	0.297	0.064
	memory	4.6MB	3.4MB	3.3MB
Vc	cycles	121M	836K	276K
	time (ms)	55.0	0.380	0.125
	memory	7.0MB	4.1MB	4.2MB

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

parameter set		key gen.	sign. gen.*	sign. verif.
Ia	cycles	9.74M	5.94M	3.06M
	time (ms)	4.43	2.70	1.39
	memory	3.5MB	2.9MB	2.7MB
IIIc	cycles	101M	66.7M	16.7M
	time (ms)	46.1	30.3	7.60
	memory	4.6MB	3.4MB	3.3MB
Vc	cycles	130M	90.3M	40.3M
	time (ms)	59.2	41.0	18.3
	memory	7.0MB	4.1MB	4.2MB

Table 9: Performance of cyclic/compressed Rainbow on Linux/Haswell (AVX2)

* decompressing from 512-bit secret key (compressed Rainbow), otherwise the same as in Table 8

6.4 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) and include overheads such as system libraries. For key generation we take the average of 10 runs; for signature generation and verification the average of 500 runs. As expected, Skylake is superior to Haswell (which is almost the same as Broadwell) in standard Rainbow. It is to be noted that our cyclic and compressed Rainbow measurements are more about memory performance, which our Haswell platform is very good at.

6.5 Note on the Verification Timings

As can be seen from Tables 5, 7 and 9, the signature verification process of cyclic and compressed Rainbow is significantly slower than that of the standard Rainbow scheme. However, we want to note that this slow down 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 slow down can be avoided nearly completely. Furthermore, additional structure in the public key might be used to speed up the evaluation of the public key further (see [14]). However, we believe that the security of Rainbow schemes with structured keys is not understood enough to propose them as a cryptographic standard. We therefore feel safer by using a public key which was generated by a cryptographically secure PRNG. However, we hope that proposing a non standard Rainbow variant here might motivate researchers to intensify their research on the security of Rainbow schemes with structured public key, too.

6.6 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, 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 the above experimental data (when looking at Rainbow instances over the same base field).

7 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 10: 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}.

7.1 General Remarks

The Rainbow signature scheme as described in Section 2.5 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).

7.2 Practical Security

In this section we analyze the security offered by the parameter sets proposed in Section 2.8.

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 8) are beyond the required levels of security.

The formulas in Section 8 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 $\mathrm{GF}(q)$ 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 8.2)
- the MinRank attack (Section 8.3)
- the HighRank attack (Section 8.4)
- the UOV attack (Section 8.5) and
- the Rainbow Band Separation (RBS) attack (Section 8.6).

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)	164.5 146.5	161.3 95.3	150.3 **86.3**	149.2 87.2	**145.0** 145.0

A collision attack against the hash function underlying the Rainbow instance Ia is at least as hard as a collision attack against SHA256 (see Section 2.6). Therefore, Rainbow instance Ia meet 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
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 IIIc is at least as hard as a collision attack against SHA384. Therefore, the Rainbow instance 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
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.

7.2.1 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 [13] it lists the proposed Rainbow instances meeting the corresponding requirements.

security category	GF(16)	GF(256)
I	Ia	-
III(IV)	-	IIIc
V(VI)	-	Vc

Table 11: Proposed Rainbow Instances and their Security Categories

7.3 Side Channel Resistance

In our implementation of the Rainbow signature scheme (see Section 5) all key dependent operations are performed in a time-constant manner. Therefore, our implementation is immune against timing attacks.

8 Analysis of Known Attacks

Known attacks against the Rainbow signature scheme include

- collision attacks against the hash function (Section 8.1)
- direct attacks (Section 8.2)
- the MinRank attack (Section 8.3)
- the HighRank attack (Section 8.4)
- The Rainbow-Band-Separation (RBS) attack (Section 8.6)
- The UOV attack (Section 8.5)

In the presence of quantum computers, one also has to consider brute force attacks accelerated by Grover's algorithm (see Section 8.7).
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.

8.1 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 $\mathcal{H}$.

8.2 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.

8.3 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

$$\text{Complexity}_{\text{MinRank; classical}} = q^{v_1+1} \cdot m \cdot \left(\frac{n^3}{3} - \frac{m^2}{6} \right) \quad (13)$$

field multiplications.

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

$$\text{Complexity}_{\text{MinRank; quantum}} = q^{\frac{v_1+1}{2}} \cdot m \cdot \left(\frac{n^3}{3} - \frac{m^2}{6} \right) \tag{14}$$

field multiplications.

There exists an alternative formulation of the MinRank attack, the so called Minors Modeling. 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 (13)). Furthermore, since we deal with a highly overdetermined system here, the MinRank attack using Minors Modeling can not be sped up by quantum techniques.

When analyzing the security of our Rainbow instances (see Section 7.2), we therefore use equations (13) and (14) to estimate the complexity of the MinRank attack.

8.4 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.

8.5 UOV - Attacks

Since Rainbow can be viewed as an extension of the well known Oil and Vinegar signature scheme [11], it can be attacked using all known UOV attacks. In particular the Reconciliation attack [8] and the UOV "Oil Subspace" attack of Kipnis and Shamir [12], of which the more serious is the latter.
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

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

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

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

field multiplications.

8.6 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.

8.7 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 [17] 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.

8.8 Security of our Rainbow Variants

Cyclic Rainbow

In [15] a number of experiments was performed which showed that the complexity of the above mentioned attacks against cyclic Rainbow is the same as against the standard Rainbow signature scheme. We furthermore do not know of any results which use the special structure of the public key of cyclic Rainbow for an attack against the scheme. We are reinforced in the security of our scheme by the fact that, due to the use of a cryptographically secure PRNG, our public key contains no visible structure. Finally, we want to mention that the key generation processes of standard and cyclic Rainbow use exactly the same subroutines and yield a one-to-one relation between the fixed parts of the public key and the central map.
We are therefore confirmed that cyclic Rainbow offers the same security as the standard Rainbow scheme and that we can use for both variants the same parameter sets.

Compressed Rainbow

The only difference of the compressed Rainbow scheme compared to cyclic Rainbow is the use of the PRNG during the signature generation process. So, if the used PRNG is cryptographically secure, the security analysis of compressed Rainbow scheme is identical to that of cyclic Rainbow.
We furthermore note that the standard Rainbow scheme, as well as many other (multivariate) schemes, uses a PRNG during the generation of the private key and therefore depend on the security of the PRNG, too. Therefore, the modification used in compressed Rainbow does not weaken the security of the scheme.

9 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 6.1).
- **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 % [15]. 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 efficient 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 and S. Bulygin. Linear Recurring Sequences for the UOV Key Generation Revisited. ICISC 2012, LNCS vol. 7839, pp. 441-455, Springer 2012.

[15] 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.

[16] 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.

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

[18] 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.