GeMSS: A Great Multivariate Short Signature

Principal submitter

This submission is from the following team, listed in alphabetical order:

- A. Casanova, CS
- J.-C. Faugère, CryptoNext, INRIA and Sorbonne University
- G. Macario-Rat, Orange
- J. Patarin, University of Versailles
- L. Perret, CryptoNext, Sorbonne University and INRIA
- J. Ryckeghem, Sorbonne University and INRIA

E-mail address: `ludovic.perret@lip6.fr`

Telephone : +33-1-44-27-88-35

Postal address:
Ludovic Perret
Sorbonne Université
LIP6 - Équipe projet INRIA/SU POLSYS
Boite courrier 169
4 place Jussieu
F-75252 Paris cedex 5, France

Auxiliary submitters: There are no auxiliary submitters. The principal submitter is the team listed above.

Inventors/developers: The inventors/developers of this submission are the same as the principal submitter. Relevant prior work is credited below where appropriate.

Owner: Same as submitter.

Signature: ×. See also printed version of "Statement by Each Submitter".

Document generated with the help of `pqskeleton` version 20170923.

Contents

1 Introduction

sparkling GeMSS *spring up from the night sky*
a dazzling splendor to ever beautify
sequined glories that verily eye smack
sparkling GeMSS *spring up from night sky*
studding the vast backdrop of black

The purpose of this document is to present GeMSS : a Great Multivariate Short Signature. As suggested by its name, GeMSS is a multivariate-based [52, 67, 27, 10, 63, 60] signature scheme producing small signatures. It has a fast verification process, and a medium/large public-key. GeMSS is in direct lineage from `QUARTZ` [59] and borrows some design rationale of the `Gui` multivariate signature scheme [28]. The former schemes are built from the *Hidden Field Equations* cryptosystem (`HFE`) [57, published in 1996] by using the so-called minus and vinegar modifiers, i.e. `HFEv-` [49]. It is fair to say that `HFE`, and its variants, are the most studied schemes in multivariate cryptography. `QUARTZ` produces signatures of 128 bits for a security level of 80 bits and was submitted to the *Nessie Ecrypt* competition [54] for public-key signatures. In contrast to many multivariate schemes, no practical attack has been reported against `QUARTZ`. This is remarkable knowing the intense activity in the cryptanalysis of multivariate schemes, e.g. [56, 50, 34, 38, 47, 46, 29, 41, 27, 10, 14, 9, 60, 65, 26]. The best known attack remains [38] that serves as a reference to set the parameters for GeMSS.

GeMSS is a faster variant of `QUARTZ` that incorporates the latest results in multivariate cryptography to reach higher security levels than `QUARTZ` whilst improving efficiency.

Acknowledgement. GeMSS has been prepared with the support of the French Programme d'Investissement d'Avenir under national project RISQ[1] P141580. For the second round, this work was also financially supported by the French Ministère des armées - Direction Générale de l'Armement.

2 General algorithm specification (part of 2.B.1)

2.1 Parameter space

The main parameters involved in GeMSS are:

- D, a positive integer that is the degree of a secret polynomial. D is such that $D = 2^i$ for $i \geq 0$, or $D = 2^i + 2^j$ for $i \neq j$, and $i, j \geq 0$,
- K, the output size in bits of the hash function,
- λ, the security level of GeMSS,
- m, number of equations in the public-key,
- nb_ite > 0, number of iterations in the verification and signature processes,

[1] https://risq.fr/?page_id=31&lang=en

- n, the degree of a field extension of $\mathbb{F}_2$,
- v, the number of vinegar variables,
- Δ, the number of minus (the number of equations in the public-key is such that is $m = n - \Delta$).

In Section 3, we specify precisely these parameters to achieve a security level $\lambda \in \{128, 192, 256\}$.

2.2 Secret-key and public-key

The public-key in G*e*MSS is a set $p_1, \ldots, p_m \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]$ of m quadratic equations in $n+v$ variables. These equations are derived from a multivariate polynomial $F \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v]$ with a specific form – as described in (1) – such that generating a signature is essentially equivalent to find the roots of F.

Secret-key. It is composed by a couple of invertible matrices $(\mathbf{S}, \mathbf{T}) \in \mathrm{GL}_{n+v}(\mathbb{F}_2) \times \mathrm{GL}_n(\mathbb{F}_2)$ and a polynomial $F \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v]$ with the following structure:

$$\sum_{\substack{0 \leqslant j < i < n \\ 2^i + 2^j \leqslant D}} A_{i,j}\, X^{2^i + 2^j} + \sum_{\substack{0 \leqslant i < n \\ 2^i \leqslant D}} \beta_i(v_1, \ldots, v_v)\, X^{2^i} + \gamma(v_1, \ldots, v_v), \tag{1}$$

where $A_{i,j} \in \mathbb{F}_{2^n}, \forall i, j, 0 \leqslant j < i < n$, each $\beta_i : \mathbb{F}_2^v \to \mathbb{F}_{2^n}$ is linear and $\gamma(v_1, \ldots, v_v) : \mathbb{F}_2^v \to \mathbb{F}_{2^n}$ is quadratic. The variables $v_1, \ldots, v_v$ are called the *vinegar variables.* We shall say that a polynomial $F \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v]$ with the form of (1) has a `HFEv`-*shape.*

Remark 1. *The particularity of a polynomial $F(X, v_1, \ldots, v_v)$ with* `HFEv`*-shape is that for any specialization of the vinegar variables the polynomial F becomes a* `HFE` *polynomial [57], i.e. univariate polynomial of the following form:*

$$\sum_{\substack{0 \leqslant j < i < n \\ 2^i + 2^j \leqslant D}} A_{i,j}\, X^{2^i + 2^j} + \sum_{\substack{0 \leqslant i < n \\ 2^i \leqslant D}} B_i\, X^{2^i} + C \in \mathbb{F}_{2^n}[X], \tag{2}$$

with $A_{i,j}, B_i, C \in \mathbb{F}_{2^n}, \forall i, j, 0 \leqslant j < i < n$.

By abuse of notation, we will call degree of F the (max) degree of its corresponding `HFE` polynomials, i.e. D.

The special structure of (1) is chosen such that its *multivariate representation* over the base field $\mathbb{F}_2$ is composed by quadratic polynomials in $\mathbb{F}_2[x_1, \ldots, x_{n+v}]$. This is due to the special exponents chosen in X that have all a binary decomposition of Hamming weight at most 2.

Let $(\theta_1, \ldots, \theta_n) \in (\mathbb{F}_{2^n})^n$ be a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. We set $\varphi : E = \sum_{k=1}^{n} e_k \cdot \theta_k \in \mathbb{F}_{2^n} \longrightarrow \varphi(E) = (e_1, \ldots, e_n) \in \mathbb{F}_2^n$.

We can now define a set of multivariate polynomials $\mathbf{f} = (f_1, \ldots, f_n) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^n$ derived from a `HFEv` polynomial $F \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v]$ by:

$$F\left(\sum_{k=1}^{n} \theta_k x_k, v_1, \ldots, v_v\right) = \sum_{k=1}^{n} \theta_k f_k\,. \tag{3}$$

To ease notations, we now identify the vinegar variables $(v_1, \ldots, v_v) = (x_{n+1}, \ldots, x_{n+v})$. Also, we shall say that the polynomials $f_1, \ldots, f_n \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]$ are the *components* of F over $\mathbb{F}_2$.

Public-key. It is given by a set of m quadratic *square-free* non-linear polynomials in $n+v$ variables over $\mathbb{F}_2$. That is, the public-key is $\mathbf{p} = (p_1, \ldots, p_m) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^m$. It is obtained from the secret-key by taking the first $m = n - \Delta$ polynomials of:

$$\Big(f_1\big((x_1, \ldots, x_{n+v})\mathbf{S}\big), \ldots, f_n\big((x_1, \ldots, x_{n+v})\mathbf{S}\big)\Big)\mathbf{T}, \tag{4}$$

and reducing it modulo the field equations, i.e. modulo $\langle x_1^2 - x_1, \ldots, x_{n+v}^2 - x_{n+v}\rangle$. We denote these polynomials by $\mathbf{p} = (p_1, \ldots, p_m) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^m$.

We summarize the public-key/secret-key generation in Algorithm (1). It takes the security parameter λ as input. As discussed in Section 8, the security level of GeMSS will be a function of D, n, v and m. In Section 3 and in Section 9, we specify precisely these parameters. Section 3 presents some parameters in order to achieve a security level $\lambda \in \{128, 192, 256\}$. In section 9, we specify some others possible parameters.

Algorithm 1 PK/SK generation in GeMSS

1: **procedure** GeMSS.KeyGen(1^λ)
2: Randomly sample $(\mathbf{S}, \mathbf{T}) \in \mathrm{GL}_{n+v}(\mathbb{F}_2) \times \mathrm{GL}_n(\mathbb{F}_2)$ ▷ This step is further detailed in Section 2.6.1.
3: Randomly sample $F \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v]$ with `HFEv`-shape of degree D ▷ This step is further detailed in Section 2.6.2.
4: $\mathsf{sk} \leftarrow (F, \mathbf{S}, \mathbf{T}) \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v] \times \mathrm{GL}_{n+v}(\mathbb{F}_2) \times \mathrm{GL}_n(\mathbb{F}_2)$
5: Compute $\mathbf{f} = (f_1, \ldots, f_n) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^n$ such that:

$$F\left(\sum_{k=1}^{n} \theta_k x_k, v_1, \ldots, v_v\right) = \sum_{k=1}^{n} \theta_k f_k$$

▷ See Section 2.6.3 for details on Step 5.

6: Compute $(p_1, \ldots, p_n) =$

$$\Big(f_1\big((x_1, \ldots, x_{n+v})\mathbf{S}\big), \ldots, f_n\big((x_1, \ldots, x_{n+v})\mathbf{S}\big)\Big)\mathbf{T} \bmod \langle x_1^2 - x_1, \ldots, x_{n+v}^2 - x_{n+v}\rangle \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^n$$

7: $\mathsf{pk} \leftarrow \mathbf{p} = (p_1, \ldots, p_m) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^m$ ▷ Take the first $m = n - \Delta$ polynomials computed in Step 6
8: **return** $(\mathsf{sk}, \mathsf{pk})$
9: **end procedure**

2.3 Signing process

The main step of the signature process requires to solve:

$$p_1(x_1, \ldots, x_{n+v}) - d_1 = 0, \ldots, p_m(x_1, \ldots, x_{n+v}) - d_m = 0. \tag{5}$$

for $\mathbf{d} = (d_1, \ldots, d_m) \in \mathbb{F}_2^m$.

To do so, we randomly sample $\mathbf{r} = (r_1, \ldots, r_{n-m}) \in \mathbb{F}_2^{n-m}$ and append it to $\mathbf{d}$. This gives $\mathbf{d}' = (\mathbf{d}, \mathbf{r}) \in \mathbb{F}_2^n$. We then compute $D' = \varphi^{-1}(\mathbf{d}' \times \mathbf{T}^{-1}) \in \mathbb{F}_{2^n}$ and try to find a root $(Z, z_1, \ldots, z_v) \in \mathbb{F}_{2^n} \times \mathbb{F}_2^v$ of the multivariate equation:

$$F(Z, z_1, \ldots, z_v) - D' = 0.$$

To solve this equation, we take advantage of the special `HFEv`-shape. That is, we randomly sample $\mathbf{v} \in \mathbb{F}_2^v$ and consider the univariate polynomial $F(X, \mathbf{v}) \in \mathbb{F}_{2^n}[X]$. This yields a `HFE` polynomial according to Remark 1. We then find the roots of the univariate equation:

$$F(X, \mathbf{v}) - D' = 0.$$

If there is a root $Z \in \mathbb{F}_{2^n}$, we return $(\varphi(Z), \mathbf{v}) \times \mathbf{S}^{-1} \in \mathbb{F}_2^{n+v}$.

A core part of the signature generation is to compute the roots of $F_{D'}(X) = F(X, \mathbf{v}) - D'$. To do so, we use the Berlekamp algorithm as described in [66, Algorithm 14.15].

Algorithm 2 Algorithm for finding the roots of a univariate polynomial

function FindRoots($F_{D'} \in \mathbb{F}_{2^n}[X]$)
  $X_n \leftarrow X^{2^n} - X \bmod F_{D'}$ ▷ This step is further detailed in Section 5.6.3
  $G \leftarrow \gcd(F_{D'}, X_n)$
  if degree$(G) > 0$ **then**
    Roots $\leftarrow$ List of all roots of G, computed by the equal-degree factorization algorithm described in [66, Section 14.3]
    return (degree(G), Roots)
  end if
  return (degree$(G), \emptyset$)
end function

The complexity of Algorithm 2 is given by the following general result:

Theorem 1 (Corollary 14.16 from [66])**.** *Let $\mathbb{F}_q$ be a finite field, and $\mathrm{M}_q(D)$ be the number of operations in $\mathbb{F}_q$ to multiply two polynomials of degree $\leq D$. Given $f \in \mathbb{F}_q[x]$ of degree D, we can find all the roots of f over $\mathbb{F}_q$ using an expected number of*

$$O\Big(\mathrm{M}_q(D)\log(D)\log(Dq)\Big)$$

or $\tilde{O}\big(D\log(q)\big)$ operations in $\mathbb{F}_q$.

For $q = 2^n$, we get that finding all the roots of a polynomial of degree D can be done in (expected) quasi-linear time, i.e.:

$$\tilde{O}(nD). \tag{6}$$

We can now present the inversion function (Algorithm 3):

Remark 2. *We sample a root at Step 12 always in the same way. First, we sort the elements of* Roots *in ascending order. We then compute* `SHA3`(D')*, and take the first* 64 *bits* H_{64} *of this hash. We view* H_{64} *as an integer, and finally return the* $(H_{64} \bmod \#\text{Roots})$*-th element in* Roots.

Algorithm 3 Inversion in GeMSS

---

function GeMSS.Inv$_{\mathbf{p}}$($\mathbf{d} \in \mathbb{F}_2^m, \mathsf{sk} = (F, \mathbf{S}, \mathbf{T}) \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v] \times \mathrm{GL}_{n+v}(\mathbb{F}_2) \times \mathrm{GL}_n(\mathbb{F}_2)$)
    repeat
        $\mathbf{r} \in_R \mathbb{F}_2^{n-m}$ ▷ The notation $\in_R$ stands for randomly sampling.
        $\mathbf{d}' \leftarrow (\mathbf{d}, \mathbf{r}) \in \mathbb{F}_2^n$
        $D' \leftarrow \varphi^{-1}(\mathbf{d}' \times \mathbf{T}^{-1}) \in \mathbb{F}_{2^n}$
        $\mathbf{v} \in_R \mathbb{F}_2^v$
        $F_{D'}(X) \leftarrow F(X, \mathbf{v}) - D'$
        $(\cdot, \mathrm{Roots}) \leftarrow \mathrm{FindRoots}(F_{D'})$
    until $\mathrm{Roots} \neq \emptyset$
    $Z \in_R \mathrm{Roots}$
    return $(\varphi(Z), \mathbf{v}) \times \mathbf{S}^{-1} \in \mathbb{F}_2^{n+v}$
end function

Let $\mathbf{d} \in \mathbb{F}_2^m$ and $\mathbf{s} \leftarrow \mathrm{Inv}_{\mathbf{p}}\big(\mathbf{d}, \mathsf{sk} = (F, \mathbf{S}, \mathbf{T})\big) \in \mathbb{F}_2^{n+v}$. By construction, we have:

$$\mathbf{p}(\mathbf{s}) = \mathbf{d}, \text{where } \mathbf{p} \text{ in the public-key associated to sk.}$$

Thus, $\mathbf{s} \in \mathbb{F}_2^{n+v}$ could be directly used as a signature for the corresponding digest $\mathbf{d} \in \mathbb{F}_2^m$. In the case of G$e$MSS, m is small enough to make the cost of simple birthday-paradox attack against the hash function more efficient that all possible attacks (as those listed in Section 8). This problem was already identified in `QUARTZ` and `Gui` [59, 22, 24, 62] who proposed to handle this issue by using the so-called *Feistel-Patarin* scheme.

The basic principle of the Feistel-Patarin scheme is to roughly iterate Algorithm 3 several times. The number of iterations is a parameter nb_ite that will be discussed in Section 6.1. We will see that we can choose nb_ite $= 4$ as in `QUARTZ` [59, 22, 24].

Algorithm 4 Signing process in GeMSS

---

procedure GeMSS.Sign($\mathbf{M} \in \{0,1\}^*, \mathsf{sk} \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v] \times \mathrm{GL}_{n+v}(\mathbb{F}_2) \times \mathrm{GL}_n(\mathbb{F}_2)$, G$e$MSS.Inv$_{\mathbf{p}}$)
    $\mathbf{H} \leftarrow \mathrm{SHA3}(\mathbf{M})$
    $\mathbf{S}_0 \leftarrow \mathbf{0} \in \mathbb{F}_2^m$
    for i from 1 to nb_ite **do**
        $\mathbf{D}_i \leftarrow$ first m bits of $\mathbf{H}$
        $(\mathbf{S}_i, \mathbf{X}_i) \leftarrow$ GeMSS.Inv$_{\mathbf{p}}(\mathbf{D}_i \oplus \mathbf{S}_{i-1})$ ▷ $\mathbf{S}_i \in \mathbb{F}_2^m$ and $\mathbf{X}_i \in \mathbb{F}_2^{n+v-m}$, $\oplus$ is the component-wise XOR
        $\mathbf{H} \leftarrow \mathrm{SHA3}(\mathbf{H})$
    end for
    return $(\mathbf{S}_{\text{nb\_ite}}, \mathbf{X}_{\text{nb\_ite}}, \ldots, \mathbf{X}_1)$ ▷ This is of size $m + \text{nb\_ite}(n+v-m) = m + \text{nb\_ite}(\Delta + v)$ bits
end procedure

2.4 Verification process

The verification process corresponding to Algorithm 4 is given in Algorithm 5.

Algorithm 5 Verification process in GeMSS

procedure GeMSS.VERIF($\mathbf{M} \in \{0,1\}^*, \text{nb\_ite} > 0, \mathsf{sm} \in \mathbb{F}_2^{m+\text{nb\_ite}(n+v-m)}, \mathsf{pk} = \mathbf{p} \in \mathbb{F}_2[x_1,\ldots,x_{n+v}]^m$)
    $\mathbf{H} \leftarrow \text{SHA3}(\mathbf{M})$
    $(\mathbf{S}_{\text{nb\_ite}}, \mathbf{X}_{\text{nb\_ite}}, \ldots, \mathbf{X}_1) \leftarrow \mathsf{sm}$
    for i from 1 to nb_ite **do**
        $\mathbf{D}_i \leftarrow$ first m bits of $\mathbf{H}$
        $\mathbf{H} \leftarrow \text{SHA3}(\mathbf{H})$
    end for
    for i from nb_ite $-$ 1 to 0 **do**
        $\mathbf{S}_i \leftarrow \mathbf{p}(\mathbf{S}_{i+1}, \mathbf{X}_{i+1}) \oplus \mathbf{D}_{i+1}$
    end for
    return VALID if $\mathbf{S}_0 = \mathbf{0}$ and INVALID otherwise.
end procedure

2.5 Data Representation

2.5.1 Compressed secret-key

The size of the secret-key can be drastically reduced. For that, we expand the secret-key from a random seed. This is classical and implies to consider a new attack: the exhaustive research of the seed. Thus, we set the size of the seed to λ bits to reach a λ-bit security level. This change increases the cost of the signing process, since the secret-key has to be generated for each operation. However, the expansion of the seed is negligible compared to the cost of the root finding. The timings are not really impacted by this modification (just slightly for RedGeMSS which has a fast signing process).

The use of a seed is controlled with the `ENABLED_SEED_SK` macro (set to 1 by default) from `config_HFE.h`. When enabled, the seed is expanded with `SHAKE`.

2.5.2 Data structure for $\mathbb{F}_2[x_1,\ldots,x_{n+v}]^m$

The first idea is to see m equations of $\mathbb{F}_2[x_1,\ldots,x_{n+v}]$ as one element in $\mathbb{F}_{2^m}[x_1,\ldots,x_{n+v}]$. The second idea is to use quadratic forms. Let $\mathbf{x} = (x_1,\ldots,x_{n+v}), C \in \mathbb{F}_{2^m}$ and $\mathbf{Q}, \mathbf{Q}' \in M_{n+v}(\mathbb{F}_{2^m})$, then a quadratic non-linear *square-free* polynomial in $\mathbb{F}_{2^m}[x_1,\ldots,x_{n+v}]$ can be written as

$$C + \mathbf{x}\mathbf{Q}'\mathbf{x}^t.$$

The coefficient $\mathbf{Q}'_{i,j}$ corresponds to the term $x_i x_j$ in the polynomial. Since $x_i^2 = x_i$, the linear term can be stored on the diagonal of $\mathbf{Q}'$.

To minimize the size, $\mathbf{Q}'$ can be transformed into a upper triangular matrix $\mathbf{Q}$. By construction, $\mathbf{Q}'_{i,j}$ and $\mathbf{Q}'_{j,i}$ are the coefficients of the same term $x_i x_j$ $(i \neq j)$. The matrix $\mathbf{Q}$ is such that:

$$\mathbf{Q}_{i,j} = \begin{cases} \mathbf{Q}'_{i,j} & \text{if } i = j \\ \mathbf{Q}'_{i,j} + \mathbf{Q}'_{j,i} & \text{if } i < j \\ 0 & \text{else.} \end{cases}$$

2.6 Implementation

We detail here some of the choices done for implementing GeMSS.

2.6.1 Generating invertible matrices

Algorithm 1 requires, at Step 2, to generate a pair of invertible matrices $(\mathbf{S}, \mathbf{T}) \in \mathrm{GL}_{n+v}(\mathbb{F}_2) \times \mathrm{GL}_n(\mathbb{F}_2)$. This problem was already discussed for `QUARTZ` [59] who presented two (natural) methods to generate invertible matrices. The first one ("*Trial and error*") sample random matrices until one is invertible. The second one, that has be chosen in `QUARTZ`, uses the so-called LU decomposition. This method has the advantage to directly return an invertible matrix. It is as follows.

- Generate a square random lower triangular L and upper triangular U matrices over $\mathbb{F}_2$, both with ones on the diagonal (to have a non-zero determinant).
- Return $L \times U$.

It is known that this method is slightly biased. A small part of the invertible matrices can not be generated with this method. For a square matrix of size n, the number of invertible triangular matrices is $2^{\sum_{i=0}^{n-1} i} = 2^{\frac{n^2-n}{2}}$. So, the number of matrices that can be generated with the LU method is $\frac{2^{n^2}}{2^n}$. This don't reduce the search space on the secret matrices sufficiently to impact the security of GeMSS.

In the code, we have implemented both generation methods. The implementation gives the possibility to switch the method with the macro `GEN_INVERTIBLE_MATRIX_LU`, which is in the file `sign_keypairHFE.c`. It is initialized to 1 by default.

The matrices $(\mathbf{S}, \mathbf{T}) \in \mathrm{GL}_{n+v}(\mathbb{F}_2) \times \mathrm{GL}_n(\mathbb{F}_2)$ are in fact only used during the generation of the public-key. After, we are only using the inverse of these matrices. So, $\mathbf{S}^{-1}$ and $\mathbf{T}^{-1}$ are computed during the generation and are stored in the secret-key.

2.6.2 Generating `HFEv` polynomials

Algorithm 1 requires, at Step 3, to generate a polynomial $F \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v]$ with `HFEv`-shape of degree D. The polynomial F can be seen as a polynomial in X whose coefficients are in $\mathbb{F}_{2^n}[v_1, \ldots, v_v]$. We store and randomly generate the non-zero exponents of F.

The polynomial F is chosen monic and so the leading coefficient is not stored. This choice makes easier the root finding part (Algorithm 2).

2.6.3 Generating the components of a `HFEv` polynomial

We detail here how to obtain the multivariate polynomials $\mathbf{f} = (f_1, \ldots, f_n) \in (\mathbb{F}_2[x_1, \ldots, x_{n+v}])^n$ from a `HFEv` polynomial $F \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v]$ such that $\sum_{k=1}^{n} \theta_k f_k$. The principle is to symbolically compute $F\left(\sum_{k=1}^{n} \theta_k x_k, v_1, \ldots, v_v\right) \in \mathbb{F}_{2^n}[x_1, \ldots, x_{n+v}]$. In the implementation, the basis

$(\theta_1, \ldots, \theta_n) \in (\mathbb{F}_{2^n})^n$ is the canonical basis of $\mathbb{F}_{2^n}$.

The polynomial F can be seen as a polynomial in X whose coefficients are in $\mathbb{F}_{2^n}[v_1, \ldots, v_v]$. We first consider terms of the form X^{2^i}. Clearly, $(\sum_{i=k}^{n} \theta_k x_k)^{2^i} = (\sum_{k=1}^{n} \theta_k^{2^i} x_k)$. We then get linear terms involved in the $f_1, \ldots, f_n$. It is the same idea for a term of the form $X^{2^i+2^j}$. We get the quadratic terms in the f_k's by $X^{2^i} X^{2^j} = (\sum_{k=1}^{n} \theta_k^{2^i} x_k) \times (\sum_{k=1}^{n} \theta_k^{2^j} x_k)$.

Since the beginning of the second round, this method is only used for the reference implementation. For the other implementations, we use the method described in [40, Section 4.1].

2.6.4 Generation of the public-key $\mathsf{pk} = \mathbf{p} \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^m$

According to Section 2.5.2, $\mathbf{f}$ is stored as $C + \mathbf{x}\mathbf{Q}\mathbf{x}^t \in \mathbb{F}_{2^n}[x_1, \ldots, x_{n+v}]$. We first compute $\big(f_1\left((x_1, \ldots, x_{n+v})\,\mathbf{S}\right), \ldots, f_n\left((x_1, \ldots, x_{n+v})\,\mathbf{S}\right)\big)$ (Step 6, Algorithm 1) with our representation. To do so, we just replace $\mathbf{x}$ by $\mathbf{x}\ \mathbf{S}$. The linear change of variables by $\mathbf{S}$ can be represented as:

$$C + \mathbf{x}\mathbf{Q'}\mathbf{x}^t \in \mathbb{F}_{2^n}[x_1, \ldots, x_{n+v}]$$

with $\mathbf{Q'} = \mathbf{S}\mathbf{Q}\mathbf{S}^{\mathbf{t}}$.

We then symmetrize the matrix $\mathbf{Q'}$ as in Section 2.5.2 to get an upper triangular matrix $\mathbf{Q''}$.

To obtain the public-key, we now need to perform linear combinations with the matrix $\mathbf{T}$. With our representation, this is equivalent to apply $\mathbf{T}$ to each coefficient to obtain the public-key in the form:

$$C_{\mathsf{pk}} + (\mathbf{x}\mathbf{Q}_{\mathsf{pk}}\mathbf{x}^t),$$

with $C_{\mathsf{pk}} \in \mathbb{F}_{2^m}$ and $\mathbf{Q}_{\mathsf{pk}} \in M_{n+v}(\mathbb{F}_{2^m})$.

In this form, the evaluation of the public-key is reduced to a matrix-vector and vector-vector products in $\mathbb{F}_{2^m}$. However, the practice use of this representation is not optimal in memory when m is not a multiple of 8. So, we must pack the bits of the public-key.

2.6.5 Packed representation of the public-key

The proposed implementation for the second round does not reached the theoretical size of the public-key. We solve this problem in our new implementation. We use a public-key format allowing to pack the bits of the public-key, while maintaining a fast use during the verifying process. On one hand, we save up to 18% of the public-key size. On the other hand, the verifying process is slightly slower (up to 31%). This change does not impact the security.

This format is based on the so-called "hybrid representation" [40]. Let $m = 8 \times k + r$ be the Euclidean division of m by 8. We store the $8k$ first equations with the monomial representation, then we store the r last equations one by one. This process is illustrated by Figure 1. Firstly, we pack the coefficients of the $8k$ first equations monomial by monomial. This corresponds to take the vertical rectangles from left to right, then to take coefficients from up to down. Secondly, we pack the coefficients of each of the r last equations. This corresponds to take the horizontal rectangles from up to down, then to take coefficients from left to right.

$$
\begin{array}{l}
c^{(1)} + p^{(1)}_{1,1}x_1^2 + p^{(1)}_{1,2}x_1x_2 + p^{(1)}_{1,3}x_1x_3 + p^{(1)}_{2,2}x_2^2 + p^{(1)}_{2,3}x_2x_3 + p^{(1)}_{3,3}x_3^2 \\
c^{(2)} + p^{(2)}_{1,1}x_1^2 + p^{(2)}_{1,2}x_1x_2 + p^{(2)}_{1,3}x_1x_3 + p^{(2)}_{2,2}x_2^2 + p^{(2)}_{2,3}x_2x_3 + p^{(2)}_{3,3}x_3^2 \\
c^{(3)} + p^{(3)}_{1,1}x_1^2 + p^{(3)}_{1,2}x_1x_2 + p^{(3)}_{1,3}x_1x_3 + p^{(3)}_{2,2}x_2^2 + p^{(3)}_{2,3}x_2x_3 + p^{(3)}_{3,3}x_3^2 \\
c^{(4)} + p^{(4)}_{1,1}x_1^2 + p^{(4)}_{1,2}x_1x_2 + p^{(4)}_{1,3}x_1x_3 + p^{(4)}_{2,2}x_2^2 + p^{(4)}_{2,3}x_2x_3 + p^{(4)}_{3,3}x_3^2 \\
c^{(5)} + p^{(5)}_{1,1}x_1^2 + p^{(5)}_{1,2}x_1x_2 + p^{(5)}_{1,3}x_1x_3 + p^{(5)}_{2,2}x_2^2 + p^{(5)}_{2,3}x_2x_3 + p^{(5)}_{3,3}x_3^2 \\
c^{(6)} + p^{(6)}_{1,1}x_1^2 + p^{(6)}_{1,2}x_1x_2 + p^{(6)}_{1,3}x_1x_3 + p^{(6)}_{2,2}x_2^2 + p^{(6)}_{2,3}x_2x_3 + p^{(6)}_{3,3}x_3^2 \\
c^{(7)} + p^{(7)}_{1,1}x_1^2 + p^{(7)}_{1,2}x_1x_2 + p^{(7)}_{1,3}x_1x_3 + p^{(7)}_{2,2}x_2^2 + p^{(7)}_{2,3}x_2x_3 + p^{(7)}_{3,3}x_3^2 \\
c^{(8)} + p^{(8)}_{1,1}x_1^2 + p^{(8)}_{1,2}x_1x_2 + p^{(8)}_{1,3}x_1x_3 + p^{(8)}_{2,2}x_2^2 + p^{(8)}_{2,3}x_2x_3 + p^{(8)}_{3,3}x_3^2 \\
\boxed{c^{(9)} + p^{(9)}_{1,1}x_1^2 + p^{(9)}_{1,2}x_1x_2 + p^{(9)}_{2,2}x_2^2 + p^{(9)}_{1,3}x_1x_3 + p^{(9)}_{2,3}x_2x_3 + p^{(9)}_{3,3}x_3^2} \\
\boxed{c^{(10)} + p^{(10)}_{1,1}x_1^2 + p^{(10)}_{1,2}x_1x_2 + p^{(10)}_{2,2}x_2^2 + p^{(10)}_{1,3}x_1x_3 + p^{(10)}_{2,3}x_2x_3 + p^{(10)}_{3,3}x_3^2}
\end{array}
$$

Figure 1: Example of hybrid represention of a multivariate quadratic system with 10 equations and 3 variables. Each row corresponds to one equation, and the $c^{(k)}$ and $p^{(k)}_{i,j}$ are in $\mathbb{F}_2$.

Our aim is to decrease the cost to unpack the bits of the public-key during the verifying process. With our format, a big part of the public-key uses the monomial representation. At the beginning of the second round, this representation was used to store the m equations (instead of $8k$ equations). So, the evaluation of the $8k$ first equations is performed as efficiently as before. They do not require to be unpacked. This implies that only the r last equations generate an additional cost, which is slight ($r \leq 7$ is small compared to $8k$). These equations can be evaluated packed, but when nb_ite > 1, to unpack them permits to accelerate the evaluation (which is repeated nb_ite times).

Implementation details

An important point in our implementation is the memory alignment. All used data has to be aligned on bytes. This permits to have more simple and more efficient implementations. In the previous implementation, we used a zero padding when necessary. However, this implied that the theoretical size was not reached.

Firstly, the $8k$ first equations are stored without loss. Since for each monomial, $8k$ coefficients in $\mathbb{F}_2$ are packed, we obtain that k bytes are required to store them. So, we do not require padding to align data on bytes. The monomials are stored in the graded lexicographic order (as on Figure 1). Secondly, the r last equations are stored in the graded reverse lexicographic order (as on Figure 1). Each equation requires to store $N = \frac{(n+v)(n+v+1)}{2}$ elements of $\mathbb{F}_2$. The alignement of the equations requires to use a zero padding when N is not multiple of 8. In this case, the padding size is $N_p = 8 - (N \bmod 8)$ bits. We solve this problem by using the $(r-1)N_p$ last bits of the last equation to fill the paddings of the $(r-1)$ other equations. In particular, we take these last bits by pack of N_p, and the ℓ-th pack is used to fill the padding of the $(8k+\ell)$-th equation. For example, on Figure 1, the 9-th equation contains 7 coefficients. So, with our process, we would remove $p_{3,3}^{(10)}$ from the 10-th equation to store it just after $p_{3,3}^{(9)}$. Thus, the 9-th equation would be aligned on 8 bits.

3 List of parameter sets (part of 2.B.1)

Following the analysis of Section 8, we propose several parameters for 128, 192 and 256 bits of classical security. Namely, we propose three sets of parameters : G*e*MSS, BlueG*e*MSS and RedG*e*MSS. G*e*MSS corresponds to the same parameters than those proposed for the first round. This choice is conservative in term of security. As advised in [55], we also explore more aggressive choice of parameters. This leads to more efficient schemes BlueG*e*MSS and RedG*e*MSS (especially, regarding the signing timings). The parameters are extracted from Section 8.6 where we propose a rather exhaustive choice of possible parameters and trade-offs between public-key size, signature size and efficiency (we use the methodology proposed in 8.6 to derive all the parameters).

3.1 Parameter sets for a security of 2^{128}

For RedG*e*MSS128, we choose nb_ite $= 4$, $\Delta = 15, v = 15$ and $m = 162$. This gives $n = 177, n+v = 192$, $D = 17$, $\lambda = 128$ and $K = 256$. In the reference implementation, the extension field is defined as $\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[X]}{X^n+X^8+1}$.

This gives a public-key of 375.21 KBytes, a signature of 282 bits, a time to sign of 2.33 MC and 141 KC to verify (Section 9.6).

For BlueG*e*MSS128, we choose nb_ite $= 4$, $\Delta = 13, v = 14$ and $m = 162$. This gives $n = 175, n+v = 189$, $D = 129$, $\lambda = 128$ and $K = 256$. In the reference implementation, the extension field is defined as $\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[X]}{X^n+X^{16}+1}$.

This gives a public-key of 363.61 KBytes, a signature of 270 bits, a time to sign of 81.3 MC and 136 KC to verify (Section 9.6).

For G*e*MSS128, we choose nb_ite $= 4$, $\Delta = 12, v = 12$ and $m = 162$. This gives $n = 174, n+v = 186$, $D = 513$, $\lambda = 128$ and $K = 256$. In the reference implementation, the extension field is defined as $\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[X]}{X^n+X^{13}+1}$.

This gives a public-key of 352.19 KBytes, a signature of 258 bits, a time to sign of 531 MC and 106 KC to verify (Section 9.6).

We summarize the parameters in the table below.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)	$\lvert pk\rvert$ (KB)	$\lvert sk\rvert$ (bits)	sign (bits)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	38.7	531	106	352.19	128	258
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	39.2	81.3	136	363.61	128	270
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	39.5	2.33	141	375.21	128	282

3.2 Parameter sets for a security of 2^{192}

For RedGeMSS192, we choose $\text{nb\_ite} = 4$, $\Delta = 23, v = 25$ and $m = 243$. This gives $n = 266, n+v = 291$, $D = 17$, $\lambda = 192$ and $K = 384$. In the reference implementation, the extension field is defined as $\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[X]}{X^n+X^{47}+1}$.

This gives a public-key of 1290.54 KBytes, a signature of 435 bits, a time to sign of 5.97 MC and 334 KC to verify (Section 9.6).

For BlueGeMSS192, we choose $\text{nb\_ite} = 4$, $\Delta = 22, v = 23$ and $m = 243$. This gives $n = 265, n+v = 288$, $D = 129$, $\lambda = 192$ and $K = 384$. In the reference implementation, the extension field is defined as $\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[X]}{X^n+X^{42}+1}$.

This gives a public-key of 1264.12 KBytes, a signature of 423 bits, a time to sign of 252 MC and 325 KC to verify (Section 9.6).

For GeMSS192, we choose $\text{nb\_ite} = 4$, $\Delta = 22, v = 20$ and $m = 243$. This gives $n = 265, n+v = 285$, $D = 513$, $\lambda = 192$ and $K = 384$. In the reference implementation, the extension field is defined as $\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[X]}{X^n+X^{42}+1}$.

This gives a public-key of 1237.96 KBytes, a signature of 411 bits and a time to sign of 1800 MC and 304 KC to verify (Section 9.6).

We summarize the parameters in the table below.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)	$\lvert pk\rvert$ (KB)	$\lvert sk\rvert$ (bits)	sign (bits)
GeMSS192	$(192, 513, 265, 22, 20, 4)$	175	1800	304	1237.96	192	411
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	174	252	325	1264.12	192	423
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	173	5.97	334	1290.54	192	435

3.3 Parameter sets for a security of 2^{256}

For RedGeMSS256, we choose $\text{nb\_ite} = 4$, $\Delta = 34, v = 35$ and $m = 324$. This gives $n = 358, n+v = 393$, $D = 17$, $\lambda = 256$ and $K = 512$. In the reference implementation, the extension field is defined as $\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[X]}{X^n+X^{57}+1}$.

This gives a public-key of 3135.59 KBytes, a signature of 600 bits, a time to sign of 9.82 MC and 704 KC to verify (Section 9.6).

For BlueGeMSS256, we choose $\text{nb\_ite} = 4$, $\Delta = 34, v = 32$ and $m = 324$. This gives $n = 358, n+v = 390$, $D = 129$, $\lambda = 256$ and $K = 512$. In the reference implementation, the extension field is defined as $\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[X]}{X^n+X^{57}+1}$.

This gives a public-key of 3087.96 KBytes, a signature of 588 bits, a time to sign of 399 MC and

684 KC to verify (Section 9.6).

For GeMSS256, we choose nb_ite $= 4, \Delta = 30, v = 33$ and $m = 324$. This gives $n = 354, n+v = 387$, $D = 513$, $\lambda = 256$ and $K = 512$. In the reference implementation, the extension field is defined as $\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[X]}{X^n+X^{99}+1}$.

This gives a public-key of 3040.70 KBytes, a signature of 576 bits, a time to sign of 3020 MC and 678 KC to verify (Section 9.6).

We summarize the parameters in the table below.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)	$\lvert pk\rvert$ (KB)	$\lvert sk\rvert$ (bits)	sign (bits)
GeMSS256	$(256, 513, 354, 30, 33, 4)$	530	3020	678	3040.70	256	576
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	530	399	684	3087.96	256	588
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	534	9.82	704	3135.59	256	600

4 Design rationale (part of 2.B.1)

A multivariate scheme. The first design rational of GeMSS is to construct a signature scheme producing short signatures. It is well known that multivariate cryptography [67, 10, 27] provides the schemes with the smallest signatures among all post-quantum schemes. Multivariate-based signature schemes are even competitive with ECC-based, pre-quantum, signature schemes (see, for example [11, 53]). This explains the choice of a multivariate cryptosystem for GeMSS.

A HFE-based scheme. HFE [57] is probably the most popular multivariate cryptosystem. Its security has been extensively studied since more than 20 years. The complexity of the best known attacks against HFE are all exponential in $O\big(\log_2(D)\big)$, where D is the degree of the secret univariate polynomial. When D is too small, then HFE can be broken, e.g. [50, 38, 9]. In contrast, solving HFE is NP-Hard when $D = O(2^n)$ [50]. However, the complexity of the signature generation – that requires finding the roots of a univariate polynomial – is quasi-linear in D (Theorem 1). All in all, there is essentially one parameter, the degree D of the univariate secret polynomial, which governs the security and efficiency of HFE. The design challenge in HFE is to find a proper trade-off between efficiency and security.

Variants of HFE. A fundamental element in the design of secure signature schemes based on HFE is the introduction of perturbations. These creates many *variants* of the scheme. Classical perturbations include the *minus modifier* (HFE-,[57]) and the *vinegar modifier* (HFEv, [49, 59]). Typically, QUARTZ is a HFEv- signature scheme where $D = 129, q = 2, n = 103, 4$ vinegar variables and 3 equations removed. The resistance, up to know, of QUARTZ against all known attacks illustrates that minus and vinegar variants permit to indeed strengthen the security of a HFE-based signature. A *nude* HFE, i.e. without any perturbation, with $D = 129$ and $n = 103$ would be insecure whilst no practical attack against QUARTZ has been reported in the literature. The best known attack is [38] that serves as a reference to set the parameters for GeMSS. Besides, [26] gave new insights on how to choose the vinegar and minus modifiers.

QUARTZ has the reputation to be solid but with a rather slow signature generation process. The authors of [59] reported a signature generation process taking about a minute. Today, the same parameters will take less than one hundred milliseconds. This is partly due to the technological progresses on the speed of processors. In fact, it is mostly due to a deeper understanding on algorithms finding the roots of univariate polynomials. This is further detailed in [40, 66].

Large set of parameters. We propose a general methodology to derive parameters. This permits to derive a large selection of parameters with various trade-offs between sizes and efficiency.

5 Detailed performance analysis (2.B.2)

5.1 Experimental Platform

Computer	Processor	Frequency	Max freq.	Architecture
LaptopS	Intel(R) Core(TM) i7-6600U CPU	2.60 GHz	3.40 GHz	Skylake
ServerH	Intel(R) Xeon(R) CPU E3-1275 v3	3.50 GHz	3.90 GHz	Haswell

Table 1: Processors.

<table>
<tr><td>Computer</td><td>OS</td><td>RAM</td><td>L1d</td><td>L1i</td><td>L2</td><td>L3</td></tr>
<tr><td>LaptopS</td><td>Ubuntu 16.04.5 LTS</td><td rowspan="2">32 GB</td><td rowspan="2">32 KB</td><td rowspan="2">32 KB</td><td rowspan="2">256 KB</td><td>4096 KB</td></tr>
<tr><td>ServerH</td><td>CentOS Linux 7 (Core)</td><td>8192 KB</td></tr>
</table>

Table 2: OS and Memory.

The measurements used one core of the CPU, and the reference implementation was compiled with `gcc -O2 -msse2 -msse3 -mssse3 -msse4.1 -mpclmul`. The SIMD is enabled only to inline the (potential) vector multiplication functions from the `gf2x` library[2]. The reference implementation does not exploit these instructions sets. For the optimized and additional implementations, the code was compiled with `gcc -O4 -mavx2 -mpclmul -mpopcnt -funroll-loops`. Turbo Boost and Enhanced Intel Speedstep Technology are disabled to have more accurate measurements

5.2 Third-party open source library

For all implementations, we have used the SHA-3 and SHAKE functions from the Extended Keccak Code Package[3]. The HFE-based schemes require to use arithmetic in $\mathbb{F}_{2^n}[X]$. In particular, the multiplication in $\mathbb{F}_{2^n}$ is the most critical operation. In the optimized and the additional implementations, we have implemented this operation by using the intel PCLMULQDQ intrinsic instruction. This instruction computes the product of two binary polynomials such that their degree is strictly less than 64. In the reference implementation, we use the fast multiplications of binary polynomials

[2] http://gf2x.gforge.inria.fr/

[3] https://keccak.team/

implemented in the `gf2x` library. In all implementations, the use of the `gf2x` library can be enabled (or disabled) by setting to 1 (or 0) the `ENABLED_GF2X` macro from `arch.h`.

5.3 Time

The following measurements are for `sign`. For signature, it signs/verifies a document of 32 bytes. For the measures, it runs a number of tests such that the global used time is greater than 1 second, and the global time is divided by the number of tests. For the signature, the lower bound of the number of tests is 256. The times of the signing process are unstable, since it depends on the probability to find a root of a univariate polynomial. So, we have taken a large number of signature.

5.3.1 Reference implementation

For the second round, we had removed the use of `NTL` in the optimized and additional implementations. This allowed to remove the use of `C++` in the implementation. The code is easier to use, more portable and more standalone. In our new implementation, we have also removed `NTL` from the reference implementation. However, the performance of the multiplication in $\mathbb{F}_2[x]$ is crucial for GeMSS. The latter was performed by `NTL`. So, we propose to switch to the `gf2x` library, which is specialized in multiplication in $\mathbb{F}_2[x]$.

These choices explain the new performances summarized in Table 3. The verifying process is more than 100 times faster, whereas the keypair generation is 13 times faster. The performance of the signing process depends on D. Indeed, `NTL` uses classical modular reductions when $D = 17$, whereas fast modular reductions are used for $D = 129$ and $D = 513$. The fast modular reduction is slower than the classical method when the input is a sparse `HFE` polynomial. So, we conclude that the vector arithmetic from `NTL` is faster than the vector multiplication from `gf2x` coupled to our reference arithmetic (without vector instructions).

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	145 / ×13	2730 / ×2.5	211 / ×140
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	118 / ×13	530 / ×1.46	228 / ×130
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	91.1 / ×13	52 / ×0.34	239 / ×110
GeMSS192	$(192, 513, 265, 22, 20, 4)$	619 / ×13	6510 / ×2.3	585 / ×150
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	520 / ×13	1290 / ×0.99	592 / ×150
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	423 / ×14	126 / ×0.22	627 / ×120
GeMSS256	$(256, 513, 354, 30, 33, 4)$	1660 / ×12	10500 / ×2.4	1160 / ×150
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	1510 / ×13	2080 / ×0.79	1190 / ×150
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	1310 / ×14	203 / ×0.18	1190 / ×120

Table 3: Performance of the reference implementation, followed by the speed-up between the new and the previous implementation. We use a Skylake processor (LaptopS). MC (resp. KC) stands for Mega (resp. Kilo) Cycles. The results have three significant digits. For example, $\boxed{145\ /\ \times 13}$ means a performance of 145 MC with the new code, and a performance of $145 \times 13 = 1880$ MC with the old code.

5.3.2 Optimized (Haswell) implementation

Since the original submission of the second round, the verifying process is between 3 and 10% slower. This is due to the fact that the public-key is stored with a packed representation. The signing process is up to 43% faster, since we have adapted the multiplication and squaring in $\mathbb{F}_2[x]$ for the Haswell processors. This counterbalances the slight cost of the secret-key decompression. The new arithmetic in $\mathbb{F}_2[x]$ improves slightly the keypair generation.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)
G*e*MSS128	$(128, 513, 174, 12, 12, 4)$	51.6 / ×1.01	1240 / ×0.98	163 / ×0.92
BlueG*e*MSS128	$(128, 129, 175, 13, 14, 4)$	52.1 / ×1.02	198 / ×1.02	170 / ×0.93
RedG*e*MSS128	$(128, 17, 177, 15, 15, 4)$	52.4 / ×1.06	5.72 / ×0.97	178 / ×0.91
G*e*MSS192	$(192, 513, 265, 22, 20, 4)$	270 / ×1.01	3320 / ×1.08	459 / ×0.96
BlueG*e*MSS192	$(192, 129, 265, 22, 23, 4)$	268 / ×1.07	481 / ×1.09	468 / ×0.94
RedG*e*MSS192	$(192, 17, 266, 23, 25, 4)$	264 / ×1.03	13.7 / ×1.01	474 / ×0.96
G*e*MSS256	$(256, 513, 354, 30, 33, 4)$	814 / ×1.04	5380 / ×1.32	973 / ×0.97
BlueG*e*MSS256	$(256, 129, 358, 34, 32, 4)$	810 / ×1.08	733 / ×1.43	989 / ×0.97
RedG*e*MSS256	$(256, 17, 358, 34, 35, 4)$	805 / ×1.07	22.1 / ×1.17	1010 / ×0.97

Table 4: Performance of the optimized implementation, followed by the speed-up between the new and the previous implementation. We use a Haswell processor (ServerH). MC (resp. KC) stands for Mega (resp. Kilo) Cycles. The results have three significant digits. For example, $\boxed{163\ /\ \times 0.92}$ means a performance of 163 KC with the new code, and a performance of $163 \times 0.92 = 150$ KC with the old code.

5.3.3 Additional (Skylake) implementation

The additional and the optimized implementations are based on the same implementation. We have only set the macro `PROC_SKYLAKE` to 1, whereas in the optimized implementation, we set the macro `PROC_HASWELL` to 1. This macro impacts mainly the multiplication in $\mathbb{F}_{2^n}$. Since the original submission of the second round, the verifying process is between 11 and 16% slower, for the same reason as before. The signing process is up to 17% slower, since the secret-key must be decompressed. The keypair generation is slightly slower because the secret-key must be decompressed and the public-key must be packed. Finally, the performance is not really impacted by our new updates, whereas keys size is smaller.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	52.6 / ×0.97	1040 / ×0.9	164 / ×0.89
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	53.8 / ×0.97	164 / ×0.97	176 / ×0.88
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	54.3 / ×0.98	5.24 / ×0.88	185 / ×0.86
GeMSS192	$(192, 513, 265, 22, 20, 4)$	275 / ×0.96	2960 / ×0.98	501 / ×0.87
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	278 / ×0.96	448 / ×0.96	512 / ×0.86
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	277 / ×0.96	13.1 / ×0.9	518 / ×0.87
GeMSS256	$(256, 513, 354, 30, 33, 4)$	916 / ×0.95	4940 / ×0.98	1120 / ×0.91
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	923 / ×0.96	653 / ×1.06	1140 / ×0.89
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	921 / ×0.97	21.4 / ×0.86	1170 / ×0.9

Table 5: Performance of the additional implementation, followed by the speed-up between the new and the previous implementation. We use a Skylake processor (LaptopS). MC (resp. KC) stands for Mega (resp. Kilo) Cycles. The results have three significant digits. For example, $\boxed{164\ /\ \times 0.89}$ means a performance of 164 KC with the new code, and a performance of $164 \times 0.89 = 146$ KC with the old code.

5.3.4 `MQsoft`

`MQsoft` [40, 1] is a new efficient library in `C` for `HFE`-based schemes such as GeMSS, `Gui` and DualModeMS. In [40], we have improved the complexity of several fundamental building blocks for such schemes and improved the protection against timing attacks. This gives the best implementation of the GeMSS family. We give here the times with the latest version of `MQsoft` [40] that uses `sse2`, `ssse3` and the `avx2` instructions sets to be faster. Since the original submission of the second round, the verifying process is between 17 and 31% slower, for the same reason as before. The signing process is between 20 and 41% faster, thanks to some optimizations. The keypair generation is not impacted.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	38.7 / ×0.99	531 / ×1.41	106 / ×0.77
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	39.2 / ×1.00	81.3 / ×1.3	136 / ×0.82
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	39.5 / ×0.99	2.33 / ×1.2	141 / ×0.77
GeMSS192	$(192, 513, 265, 22, 20, 4)$	175 / ×1.00	1800 / ×1.29	304 / ×0.79
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	174 / ×0.99	252 / ×1.31	325 / ×0.78
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	173 / ×0.99	5.97 / ×1.4	334 / ×0.76
GeMSS256	$(256, 513, 354, 30, 33, 4)$	530 / ×1.00	3020 / ×1.21	678 / ×0.83
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	530 / ×1.00	399 / ×1.37	684 / ×0.85
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	534 / ×0.98	9.82 / ×1.31	704 / ×0.84

Table 6: Performance of `MQsoft`, followed by the speed-up between the new and the previous implementation. We use a Skylake processor (LaptopS). MC (resp. KC) stands for Mega (resp. Kilo) Cycles. The results have three significant digits. For example, $\boxed{106\ /\ \times 0.77}$ means a performance of 106 KC with the new code, and a performance of $106 \times 0.77 = 82$ KC with the old code.

5.4 Space

In Table 7, we provide the updated sizes of the public-key, secret-key and signature. From now on, the implementation optimizes the sizes. The theoretical and practical sizes are the same. Since the secret-key is generated from a seed (Section 2.5.1), the secret-key is very small: just several hundreds of bits. In contrast, the decompressed secret-key size is between 10 and 80 KB. For the public-key, we have decreased the practical size (Section 2.6.5). We save 18% for $\lambda = 128$, 5% for $\lambda = 192$ and 0.2% for $\lambda = 256$ (the latter was already optimized since the beginning of the second round).

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	$\lvert pk\rvert$ (KB)	$\lvert sk\rvert$ (B)	sign (B)
G*e*MSS128	$(128, 513, 174, 12, 12, 4)$	352.188	16	32.25
BlueG*e*MSS128	$(128, 129, 175, 13, 14, 4)$	363.609	16	33.75
RedG*e*MSS128	$(128, 17, 177, 15, 15, 4)$	375.21225	16	35.25
G*e*MSS192	$(192, 513, 265, 22, 20, 4)$	1237.9635	24	51.375
BlueG*e*MSS192	$(192, 129, 265, 22, 23, 4)$	1264.116375	24	52.875
RedG*e*MSS192	$(192, 17, 266, 23, 25, 4)$	1290.542625	24	54.375
G*e*MSS256	$(256, 513, 354, 30, 33, 4)$	3040.6995	32	72
BlueG*e*MSS256	$(256, 129, 358, 34, 32, 4)$	3087.963	32	73.5
RedG*e*MSS256	$(256, 17, 358, 34, 35, 4)$	3135.591	32	75

Table 7: Memory cost. 1 KB is 1000 bytes.

5.5 How parameters affect performance

Signature generation is mainly affected by n and the degree D of the secret univariate polynomial. According to Theorem 1, we can find the roots of $F \in \mathbb{F}_{2^n}[X]$ in $\tilde{O}\big(n\,D\big)$ binary operations. So, n and D are the main parameters which influence the efficiency. In Sec. 8, we will see how to choose these parameters in function of the security parameter.

5.6 Optimizations

The optimized and additional implementations modify the order of computations to have the best possible contiguity, and in this way avoids a maximum of miss in the cache. The implementation avoids to store useless null coefficients (for example, for a triangular matrix), and every data are stored in unidimensional tabular of words.

5.6.1 Improvement of the arithmetic in $\mathbb{F}_{2^n}$

The multiplication in $\mathbb{F}_{2^n}$ is the most expensive part of G*e*MSS: the generation of the public-key/secret-key requires $O\big(n\log_2(D)(n+v+\log_2(D))\big)$ field multiplications, and the signature requires $\tilde{O}\big(n\,D\big)$ field multiplications.

The additional implementation uses the schoolbook multiplication, whereas the optimized implementation uses the Karatsuba algorithm. Both use the `_mm_clmulepi64_si128` intrinsic for the basis case. This intrinsic calls the `PCLMULQDQ` instruction.

The squaring in $\mathbb{F}_{2^n}$ is important in the signature generation. Indeed, the computation of $(X^{2^n} - X) \mod F$ (Algorithm 2) requires $O(nD)$ squaring. The squaring consists just to interleave a zero bit between each bit of the input. To do this, the additional implementation uses several times the intrinsic `_mm_clmulepi64_si128`, which computes directly the squaring of a 64-bit element. The optimized implementation uses mainly the `VPSHUFB` instruction from the `AVX2` instructions sets.

5.6.2 Evaluation of the public-key

Before our new implementation proposed during the second round, the public-key was represented in the form:

$$C_{\mathsf{pk}} + \mathbf{x}\mathbf{Q}_{\mathsf{pk}}\mathbf{x}^t,$$

with $C_{\mathsf{pk}} \in \mathbb{F}_{2^m}$ and $\mathbf{Q}_{\mathsf{pk}} \in M_{n+v}(\mathbb{F}_{2^m})$.

The optimization is to set to zero the i-th row of $\mathbf{Q}_{\mathsf{pk}}\mathbf{v}^t$ (a column vector) if the i-th component of $\mathbf{v}$ is null. We avoid a dot product for each null coefficient.

With our new implementation, only the $8k$ first equations (instead of m) are stored with the previous format (Section 2.6.5). Once unpacked, the r last equations are in the form:

$$C + \mathbf{x}\mathbf{Q}\mathbf{x}^t,$$

with $C \in \mathbb{F}_2$ and $\mathbf{Q} \in M_{n+v}(\mathbb{F}_2)$ a lower triangular matrix.

So, each equation can be evaluated with matrix-vector and vector-vector product in $\mathbb{F}_2$. The previous optimization is also applied here.

5.6.3 Computation of the Frobenius map

To compute the roots of $F_{D'} = F(X, \mathbf{v}) - D'$ (Algorithm 2) during the signature, the reference implementation uses the `FrobeniusMap` function from `NTL`. To accelerate this function, the other implementations use a `C` implementation of $(X^{2^n} - X) \mod F_{D'}$, as this:

Algorithm 6 Algorithm for the Frobenius map

function FROBENIUS_MAP($F_{D'}, n$)
  Choose a such that $2^a < \text{degree}(F_{D'})$ but $2^{a+1} \geq \text{degree}(F_{D'})$.
  $X_a \leftarrow X^{2^a}$
  for i from $a+1$ to n **do**
    $X_i \leftarrow (X_{i-1})^2$ ▷ Linearity of the Frobenius endomorphism
    $X_i \leftarrow X_i \mod F_{D'}$ ▷ We use the fact that $F_{D'}$ is monic and sparse
  end for
  return $X_n + X$
end function

The computation of the squaring is equivalent to compute the square of each coefficient, and put a null coefficient between each coefficient. Since $F_{D'}$ is monic, it is useless to multiply $F_{D'}$ by the inverse of its leading coefficient to compute the modular reduction. The fact that $F_{D'}$ is sparse avoids to load and read useless null coefficients, since just the useful coefficients are stored.

Note: during the second round, `NTL` has been removed from all implementations.

6 Expected strength (2.B.4) in general

We review in this part known results on the provable security of GeMSS. This includes the required number of iterations in the Feistel-Patarin scheme (Section 6.1) as well as the security (Section 6.2) in the sense of the existential unforgeability against adaptive chosen-message attack (EUF-CMA).

6.1 Number of iterations nb_ite in Sign and Verif

We explain here how the number of iterations $\text{nb\_ite} > 0$ has to be chosen in Algorithms 4 and 5. This follows from the analysis performed already in `QUARTZ` [59, 22].

Theorem 2 (adapted from [22])**.** *The number of iterations* nb_ite *has to be chosen such that*

$$2^{m\frac{\text{nb\_ite}}{\text{nb\_ite}+1}} \geq 2^{\lambda}.$$

We use this result to derive the number of iterations for all parameters of GeMSS.

6.2 EUF-CMA security

EUF-CMA security of `HFEv-`, over which GeMSS is designed, has been mainly investigated in [64]. The authors demonstrated that a minor, but costly, modification of GeMSS.$\text{Inv}_{\mathbf{p}}$ (Algorithm 3) permits to achieve EUF-CMA security for GeMSS. In fact, the result of [64] applies more precisely to a version of GeMSS.$\text{Inv}_{\mathbf{p}}$ where nb_ite is equal to one. In this case, the EUF-CMA security of (modified) GeMSS follows easily from [64].

We first formalize the security of GeMSS against chosen message attacks.

Definition 1 ([64])**.** *The* GeMSS *signature scheme* (GeMSS.KeyGen, GeMSS.Sign, GeMSS.Verif) *is* $\big(\epsilon(\lambda), q_s(\lambda), q_h(\lambda), t(\lambda)\big)$*-secure if there is no forger* A *who takes as input a public-key* $(\cdot, \mathsf{pk}_{\mathsf{GeMSS}}) \leftarrow$ GeMSS.KeyGen() *and with at most* $q_h(\lambda)$ *queries to the random oracle,* $q_s(\lambda)$ *queries to the signature oracle, then outputs a valid signature after* $t(\lambda)$ *steps with a probability at least* $\epsilon(\lambda)$*.*

We want to provably reduce EUF-CMA security of GeMSS to the the hardness of inverting the public-key of GeMSS. Formally:

Definition 2 ([64])**.** *We shall say that the* GeMSS *function generator* GeMSS.KeyGen *is* $\big(\epsilon(\lambda), t(\lambda)\big)$ *secure, if there is no inverting algorithm that takes* $\mathsf{pk}_{\mathsf{GeMSS}} = \mathbf{p}_{\mathsf{GeMSS}}$ *generated via*

$(\cdot, \mathsf{pk}_{\mathsf{GeMSS}}) \leftarrow \mathsf{GeMSS}.\textsc{KeyGen}(1^\lambda)$, *a challenge* $\mathbf{d} \in_R \mathbb{F}_2^m$, *and finds a preimage* $\mathbf{s} \in_R \mathbb{F}_2^{n+v}$ *such that*

$$\mathbf{p}_{\mathsf{GeMSS}}(\mathbf{s}) = \mathbf{d}.$$

after $t(\lambda)$ *steps with success probability at least* $\epsilon(\lambda)$.

Following [64], we explain now how to modify GeMSS for proving EUF-CMA security. Recall that D is degree of the secret polynomial with HFEv-shape in GeMSS. The main modification proposed by [64] is roughly to repeat D times the inversion step described in Algorithm 3.

Let ℓ be the length of a random salt. The modified inversion process is given in Algorithm 7:

Algorithm 7 Modified inversion for GeMSS

1: **procedure** $\mathsf{GeMSS}.\mathrm{Inv}^*_{\mathbf{p}}(\mathbf{d} \in \mathbb{F}_2^m, \ell \in \mathbb{N}, \mathrm{sk} = (F, \mathbf{S}, \mathbf{T}) \in \mathbb{F}_{2^n}[X, v_1, \ldots, v_v] \times \mathrm{GL}_{n+v}(\mathbb{F}_2) \times \mathrm{GL}_n(\mathbb{F}_2))$
2: $\quad\mathbf{v} \in_R \mathbb{F}_2^v$
3: $\quad$**repeat**
4: $\quad\quad\mathbf{salt} \in_R \{0,1\}^\ell$
5: $\quad\quad\mathbf{r} \leftarrow$ first $n - m$ bits of $\mathtt{SHA3}(\mathbf{d} \| \mathbf{salt})$
6: $\quad\quad\mathbf{d}' \leftarrow (\mathbf{d}, \mathbf{r}) \in \mathbb{F}_2^n$
7: $\quad\quad D' \leftarrow \varphi^{-1}(\mathbf{d}' \times \mathbf{T}^{-1}) \in \mathbb{F}_{2^n}$
8: $\quad\quad F_{D'}(X) \leftarrow F(X, \mathbf{v}) - D'$
9: $\quad\quad(\cdot, \mathrm{Roots}) \leftarrow \mathrm{FindRoots}(F_{D'})$
10: $\quad\quad u \in_R \{1, \ldots, D\}$
11: $\quad$**until** $1 \leq u \leq \#\mathrm{Roots}$
12: $\quad Z \in_R \mathrm{Roots}$
13: $\quad$**return** $(\varphi(Z), \mathbf{v}) \times \mathbf{S}^{-1} \in \mathbb{F}_2^{n+v}$
14: **end procedure**

Given Algorithm 7, we can define $\mathsf{GeMSS}.\textsc{Sign}^*$ as the signature algorithm 4 instantiated with $\mathsf{GeMSS}.\mathrm{Inv}^*_{\mathbf{p}}$ and with nb_ite $= 1$. Similarly, $\mathsf{GeMSS}.\textsc{Verif}^*$ is the verification algorithm 5 where nb_ite $= 1$.

Theorem 3 ([64]). *Let* GeMSS^* *be the signature scheme defined by* $(\mathsf{GeMSS}.\textsc{KeyGen}, \mathsf{GeMSS}.\textsc{Sign}^*, \mathsf{GeMSS}.\textsc{Verif}^*)$. *Thus, if the* GeMSS *function generator* $\mathsf{GeMSS}.\textsc{KeyGen}$ *is* (ϵ', t') *secure, then* GeMSS^* *is* (ϵ, t, q_H, q_S) *secure, with:*

$$\begin{aligned} \epsilon &= \frac{\epsilon'(q_H + q_s + 1)}{1 - (q_H + q_s)q_s 2^\ell}, \\ t &= \frac{t' - (q_H + q_s + 1)}{t_{\mathsf{GeMSS}} + O(1)} \end{aligned}$$

where t_{GeMSS} *is the time required to evaluate the public-key of* GeMSS.

There are two differences between GeMSS and GeMSS^*. First, $\mathsf{GeMSS}.\mathrm{Inv}^*_{\mathbf{p}}$ is more costly than $\mathsf{GeMSS}.\mathrm{Inv}_{\mathbf{p}}$. The expected number of calls to the root-finding step (Step 9) in $\mathsf{GeMSS}.\mathrm{Inv}^*_{\mathbf{p}}$ is $\frac{1}{1-1/e} D \approx 1.58 \times D$. In $\mathsf{GeMSS}.\mathrm{Inv}_{\mathbf{p}}$, the average number of calls to the root-finding step (Step 8) is $\frac{1}{1-1/e} \approx 1.58$.

In GeMSS, we are typically considering D between 17 and 513. For efficiency reasons, we did not incorporated this modification in our implementation.

Remark 3. *The threshold D in Step 10 corresponds to a bound on the number of roots of the univariate polynomial F at Step 9. However, F has a* `HFE`*-shape (Remark 1) and has much less roots than a random univariate polynomial of the same degree. Indeed, the roots of a* `HFE` *polynomial correspond to the zeros of a system of n boolean equations in n variables (see (3)). In [42], the authors studied the distribution of the number of zeroes of algebraic systems. In particular, a random system of n equations in n variables has exactly s solutions with probability $\frac{1}{e\,s!}$. Thus, as also mentionned [64], the threshold D in Step 10 can be theoretically much decreased without compromising the proof. The authors of [64] mentioned a value around ≈ 30 for the threshold.*

The second difference between GeMSS and GeMSS* is on the number of iterations. The treatment of [64] did not include the use of a Feistel-Patarin transform. It is an interesting open problem to formally prove EUF-CMA security when nb_ite > 0. This should probably follow from the use of Theorem 2.

All in all, the provable security results mentioned up to know only require minor modifications of the signature process without changing the underlying trapdoor. As a consequence, the security of GeMSS has to be mainly studied with respect to the hardness of inverting the public-key. This question is investigated in Section 8.

6.3 Signature failure

This analysis is essentially similar to the one performed for `QUARTZ` [59]. A failure can occurs in GeMSS.Inv$_{\mathbf{p}}$ (Algorithm 3), at Step 8, if Roots $= \emptyset$ for all $(\mathbf{r}, \mathbf{v}) \in \mathbb{F}_2^{n-m} \times \mathbb{F}_2^{v}$. The probability that Roots is empty for a given $(\mathbf{d}, \mathbf{v}) \in \mathbb{F}_2^{m} \times \mathbb{F}_2^{v}$ is $1/e$ [59, 42]. Thus, Algorithm 7 fails with probability $(\frac{1}{e})^{2^{n+v-m}}$.

Finally, GeMSS.Inv$_{\mathbf{p}}$ is called GeMSS.Sign nb_ite times. The probability of failure for GeMSS.Sign is then:

$$1-\left(1-\left(\frac{1}{e}\right)^{2^{n+v-m}}\right)^{\text{nb\_ite}}.$$

7 Expected strength (2.B.4) for each parameter set

7.1 Parameter set `sign/BlueGeMSS128`

Category 1.

7.2 Parameter set `sign/BlueGeMSS192`

Category 3.

7.3 Parameter set `sign/BlueGeMSS256`

Category 5.

7.4 Parameter set `sign/GeMSS128`

Category 1.

7.5 Parameter set `sign/GeMSS192`

Category 3.

7.6 Parameter set `sign/GeMSS256`

Category 5.

7.7 Parameter set `sign/RedGeMSS128`

Category 1.

7.8 Parameter set `sign/RedGeMSS192`

Category 3.

7.9 Parameter set `sign/RedGeMSS256`

Category 5.

8 Analysis of known attacks (2.B.5)

This part provides a summary of the main attacks against G*e*MSS. In Section 8.1, we consider direct signature forgery attacks. This includes, in particular, the analysis of known quantum attacks (Sections 8.1.2 and 8.3) and Gröbner basis attacks (Sections 8.1.2 and 8.3). In Section 8.4, we consider key-recovery attacks.

In almost all cases, the attacks reduce to solving a particular system of non-linear equations derived from the public polynomials.

8.1 Direct signature forgery attacks

The public-key of G*e*MSS is given by a set of non linear-equations $\mathbf{p} = (p_1, \ldots, p_m) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^m$. Given a digest $(d_1, \ldots, d_m) \in \mathbb{F}_2^m$, the problem of forging a signature is equivalent to solve the following system of non-linear equations:

$$p_1(x_1, \ldots, x_{n+v}) - d_1 = 0, \ldots, p_m(x_1, \ldots, x_{n+v}) - d_m = 0, x_1^2 - x_1, \ldots, x_{n+v}^2 - x_{n+v} = 0. \quad (7)$$

Stated differently, the task is to invert $\mathsf{G}e\mathsf{MSS}.\mathrm{Inv}_{\mathbf{p}}$ (Algorithm 3) without the knowledge of the secret-key sk.

In our case, the system is under-defined, i.e. $n + v > m$. As a consequence, we can randomly fix $n + v - m$ variables $\mathbf{r} = (r_1, \ldots, r_{n+v-m}) \in \mathbb{F}_2^{n+v-m}$ in (7) and try to solve for the remaining variables. Note that this is similar to the (legitimate) signature process which requires to randomly fix variables in $\mathsf{G}e\mathsf{MSS}.\mathrm{Inv}_{\mathbf{p}}$ (Steps 3 and 6 of Algorithm 3).

Thus, the problem of forging a signature reduces to solve a system of m quadratic equations in m variables over $\mathbb{F}_2$:

$$p_1(x_1, \ldots, x_m, \mathbf{r}) - d_1 = 0, \ldots, p_m(x_1, \ldots, x_m, \mathbf{r}) - d_m = 0, x_1^2 - x_1, \ldots, x_m^2 - x_m = 0. \quad (8)$$

8.1.1 Exhaustive search

In [13], the authors describe a fast exhaustive search for solving systems of boolean quadratic equations. They also provide a detailed cost analysis of their approach. To recover a solution of (8), the approach from [13] requires:

$$4\,\log_2(m)\,2^m \text{ binary operations.}$$

For the parameters of $\mathsf{G}e\mathsf{MSS}$, we obtain for example:

m	Fast exhaustive search ([13])
162	$2^{166.87}$
243	$2^{247.98}$
324	$2^{329.98}$

We always take into account this attack to derive all the parameters proposed in this document (typically, $\mathsf{BlueG}e\mathsf{MSS}$, $\mathsf{RedG}e\mathsf{MSS}$ and the parameters of Section 9). The same remark holds for all attacks described from now on.

8.1.2 Quantum exhaustive search

In [19], the authors proposed simple quantum algorithms for solving systems of quadratic boolean equations. The principle of [19] is to perform a fast quantum exhaustive search by using Grover's algorithm. [19] demonstrated that we can solve a system of $m - 1$ binary quadratic equations in $n - 1$ binary variables using $m + n + 2$ qubits and evaluating a circuit of $2^{n/2}\Big(2m(n^2 + 2n) + 1\Big)$ quantum gates. They also describe a variant using less qubits, i. e. $3 + n + \lceil \log_2(m) \rceil$ qubits, but requiring to evaluate a larger circuit, i.e. with $\approx 2 \times 2^{n/2}\Big(2m(n^2 + 2n) + 1\Big)$ quantum gates.

We can now estimate is the cost for solving the system (8). For $\mathsf{G}e\mathsf{MSS}$, the quantum attacks from [19] require for example :

m	#qbits	#quantum gates
162	328	$2^{104.56}$
162	173	$\approx 2^{105.56}$
243	490	$2^{146.8}$
243	254	$\approx 2^{146.8}$
324	652	$2^{188.54}$
324	336	$\approx 2^{189.54}$

8.2 Approximation algorithm

Recently, the authors of [51] proposed a new algorithm for solving systems of non linear equations that is faster than a direct exhaustive search. The techniques from [51] allow for the approximation of a non-linear system, as (8), by a single high-degree multivariate polynomial P with $m' < m$ variables. The polynomial P is constructed such that it vanishes on the same zeroes as the original non-linear system with high probability. We then perform an exhaustive search on P to recover, with high probability, the zeroes of the non-linear system. This leads to an algorithm for solving (8) whose asymptotic complexity is:

$$O^*\left(2^{0.8765\,m}\right).$$

The notation O^* omits polynomial factors. Anyway, we will estimate the cost of this attack by the lower bound $2^{0.8765\,m}$.

For the parameters of GeMSS, we have then:

m	Lower bound on the complexity of [51]
162	$2^{141.99}$
243	$2^{212.98}$
324	$2^{283.98}$

8.3 Gröbner bases

To date, the best methods for solving non-linear equations, including the attack system (8), utilize Gröbner bases [17, 16]. The historical method for computing such bases – known as Buchberger's algorithm – has been introduced by Buchberger in his PhD thesis [17, 16]. Many improvements on Buchberger's algorithm have been done leading – in particular – to more efficient algorithms such as the F4 and F5 algorithms of J.-C. Faugère [32, 33]. The F4 algorithm, for example, is the default algorithm for computing Gröbner bases in the computer algebra software MAGMA [12]. The F5 algorithm, which is available through the FGb [35] software[4], provides today the state-of-the-art method for computing Gröbner bases.

Besides F4 and F5, there is a large literature of algorithms computing Gröbner bases. We mention for instance `PolyBory` [15] which is a general framework to compute Gröbner basis in $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_i^2 - x_i\rangle_{1\leq i\leq n}$. It uses a specific data structure – dedicated to the Boolean ring – for computing Gröbner basis on top of a tweaked Buchberger's algorithm[5]. Another technique

[4] `http://www-polsys.lip6.fr/~jcf/FGb/index.html`
[5] `http://polybori.sourceforge.net`

proposed in cryptography is the `XL` algorithm [23]. It is now clearly established that `XL` is a special case of Gröbner basis algorithm [2]. More recently, a zoo of algorithms such as `G2V` [44], `GVW` [45], ..., flourished building on the core ideas of F4 and F5. This literature is vast and we refer to [31] for a recent survey of these algorithms.

Despite this important algorithmic literature, if is fair to say that MAGMA and FGb remain the references softwares for polynomial system solving over finite fields. We have intensively used both softwares to perform practical experiments and support our methodology to derive secure parameters (Section 8.3.3).

8.3.1 Asymptotically fast algorithms

`BooleanSolve` [7] is the fastest asymptotic algorithm for solving system of non-linear boolean equations. `BooleanSolve` is a hybrid approach that combines exhaustive search and Gröbner bases techniques. For a system with the same number of equations and variables (m), the deterministic variant of `BooleanSolve` has complexity bounded by $O(2^{0.841m})$, while a Las-Vegas variant has expected complexity

$$O(2^{0.792 \cdot m}).$$

It is mentioned in [7] that `BooleanSolve` is better than exhaustive search when $m \geq 200$. This is due to the fact that large constants are hidden in the big-O notation. As a conservative choice, we lower bound here the cost of this attack by $2^{0.792 \cdot m}$. We mention that [61] recently considered a hybrid approach against `HFEv-`. The former result also indicates that our approach is indeed conservative.

In Table 8, we report the security level of GeMSS against `BooleanSolve` (probabilistic version) for the three security levels proposed.

m	Lower bound on the cost of `BooleanSolve` ($2^{0.792 \cdot m}$)
162	$2^{128.3}$
243	$2^{192.45}$
324	$2^{256.6}$

Table 8: Security of GeMSS against `BooleanSolve`.

In fact, we have used `BooleanSolve` as the reference approach to derive the minimal number m of equation required in GeMSS.

`QuantumBooleanSolve`. In [37], the authors present a quantum version of `BooleanSolve` that takes advantages of Grover's quantum algorithm [48]. `QuantumBooleanSolve` is a Las-Vegas quantum algorithm allowing to solve a system of m boolean equations in m variables. It uses $O(n)$ qbits, requires the evaluation of, on average, $O(2^{0.462m})$ quantum gates. This complexity is obtained under certain algebraic assumptions.

In Table 9, we report the security level of GeMSS against `QuantumBooleanSolve` (probabilistic version) for the three security levels proposed.

m	Lower bound on the # quantum gates for `QuantumBooleanSolve` ($2^{0.462 \cdot m}$)
162	$2^{74.84}$
243	$2^{112.26}$
324	$2^{149.68}$

Table 9: Security of G*e*MSS against `QuantumBooleanSolve`.

Note that [8] also proposed a new (Gröbner-based) quantum algorithm for solving quadratic equations with a complexity comparable to `QuantumBooleanSolve` (we refer to [37] for further details).

8.3.2 Practically fast algorithms

The direct attack described in [34, 38] provides reference tools for evaluating the security of `HFE` and `HFEv-` against a direct message-recovery attack. This attack uses the F5 algorithm [33, 5] and has a complexity of the following general form:

$$O\big(\mathrm{poly}(m,n)^{\omega \cdot D_{\mathrm{reg}}}\big), \tag{9}$$

with $2 \leq \omega < 3$ being the so-called *linear algebra constant* [66], i.e. the smallest constant $\omega, 2 \leq \omega < 3$ such that two matrices of size $N \times N$ over a field $\mathbb{F}$ can be multiplied in $O(N^{\omega})$ arithmetic operations over $\mathbb{F}$. The best current bound is $\omega < 2.3728639$ [43]. In this part, we will always use $\omega = 2$ to evaluate the cost of Gröbner bases attacks.

The complexity (9) is exponential in the *degree of regularity* D_{reg} [3, 6, 4]. However, this degree of regularity D_{reg} can be difficult to predict in general ; as difficult than computing a Gröbner basis. Fortunately, there is a particular class of systems for which this degree can be computed efficiently and explicitly : *semi-regular sequences* [3, 6, 4]. This notion is supposed to capture the behavior of a random system of non-linear equations. In order to set the parameters for `HFE` and variants as well than for performing meaningful experiments on the degree of regularity, we can assume that no algebraic system has a degree of regularity higher than a semi-regular sequence.

In Table 10, we provide the degree of regularity of a semi-regular system of m boolean equations in m variables for various values of m.

In the case of `HFE`, the degree of regularity for solving (8) has been experimentally shown to be smaller than $\log_2(D)$ [34, 38]. This behavior has been further demonstrated in [47, 30]. In particular, [47] claims that the degree of regularity reached in `HFE` is asymptotically upper bounded by:

$$(2+\epsilon)(1-\sqrt{3/4}) \cdot \min\big(m, \log_2(D)\big), \text{ for all } \epsilon > 0. \tag{10}$$

This bound is obtained by estimating the degree of regularity of a semi-regular system of $3\lceil \log_2(D) \rceil$ quadratic equations in $2\lceil \log_2(D) \rceil$ variables. We emphasize that an asymptotic bound such as (10) is not necessarily tight for specified values of the parameters. Thus, (10) can not be directly used to derive actual parameters but still provide a meaningful asymptotic trend.

Indeed, the behavior of `HFE` algebraic systems is then much different from a semi-regular system of m boolean equations in m variables where the degree of regularity increases linearly with m. Roughly, D_{reg} grows as $\approx m/11.11$ in the semi-regular case [3, 6, 4].

m	D_{reg}
$3 \leq m \leq 8$	3
$9 \leq m \leq 15$	4
$16 \leq m \leq 23$	5
$24 \leq m \leq 31$	6
$32 \leq m \leq 40$	7
$41 \leq m \leq 48$	8
$49 \leq m \leq 57$	9
$58 \leq m \leq 66$	10
$154 \leq m \leq 163$	20
$234 \leq m \leq 243$	28
$316 \leq m \leq 325$	36

Table 10: Degree of regularity of m semi-regular boolean equations in m variables.

We report below the degree of regularity $D_{\mathrm{reg}}^{\mathrm{Exp}}$ observed in practice for `HFE`. These bounds are are only meaningful for a sufficiently large m which is given in the first column. Indeed, as we already explained, we can assume that the values from Tab. 10 are upper bounds on the degree of regularity of any algebraic system of boolean equations.

Minimal m	`HFE`(D)	$D_{\mathrm{reg}}^{\mathrm{Exp}}$
$\geqslant 3$	$3 \leq D \leq 16$	3
$\geqslant 9$	$17 \leq D \leq 128$	4
$\geqslant 16$	$129 \leq D \leq 512$	5
$\geqslant 24$	$513 \leq D \leq 4096$	6
$\geqslant 32$	$D \geq 4097$	7

Table 11: Degree of regularity in the case of `HFE` algebraic systems.

Following [38], we lower bound the complexity of F5 against `HFE`, i.e. for solving the attack system (8). The principle is to only consider the cost of performing a row-echelon computation on a full rank sub-matrix of the biggest matrix occurring in F5. At the degree of regularity, this sub-matrix has $\binom{m}{D_{\mathrm{reg}}}$ columns and (at least) $\binom{m}{D_{\mathrm{reg}}}$ rows. Thus, we can bound the complexity of a Gröbner basis computation against `HFE` by:

$$O\left(\binom{m}{D_{\mathrm{reg}}}^2\right). \tag{11}$$

This is a conservative estimate on the cost of solving (8). This represents the minimum computation that has to be done in F5. We also assumed that the linear algebra constant ω is 2; the smallest possible value.

Given a value of m, we can now deduce from (11) and Table 8, the (smallest) degree of regularity required to achieve a certain security level. These values are given in Table 12.

From Table (11), we can see that no `HFE` has a degree of regularity sufficiently large to achieve a reasonable level of security. To do so, we need to use modifiers of `HFE` for increasing the degree of regularity.

m	minimal D_{reg} required	Lower bound on the cost of a Gröbner basis as given in (11)
162	14	$2^{131.16}$
243	20	$2^{192.52}$
324	27	$2^{260.86}$

Table 12: Smallest degree of regularity required.

In particular, the practical effect of the minus and vinegar modifiers have been considered in [34, 38]. This has been further investigated in [25, 28] who presented a theoretical upper bound on the degree of regularity arising in HFEv-. Let $R = \lfloor \log_2(D-1) \rfloor + 1$, then the degree of regularity for HFEv- is bounded from above by

$$\frac{R+v+\Delta-1}{2}+2, \quad \text{when } R+\Delta \text{ is odd,} \tag{12}$$

$$\frac{R+v+\Delta}{2}+2, \quad \text{otherwise.} \tag{13}$$

We observe that degree of regularity seems to increase linearly with $(n+v-m)$. This is the sum of the modifiers : number of equations removed plus vinegar variables.

Very recently, [61] derived an experimental *lower bound* on the degree of regularity in HFEv-. The authors [61] obtained that the degree of regularity for HFEv- should be at least :

$$\left\lceil \frac{R+\Delta+v+7}{3} \right\rceil. \tag{14}$$

8.3.3 Experimental results for HFEv-

The main question in the design of GeMSS is to quantify, as precisely as possible, the effect of the modifiers on the degree of regularity. To do so, we performed experimental results on the behaviour of a direct attack against HFEv-, i.e. computing a Gröbner basis of (8). We mention that similar experiments were performed in [62].

We first consider $v = 0$, and denote by Δ the number of equation removed, i.e. $m = n - r$. According to the upper bounds (12) and (13), the degree of regularity should increase by 1 when 2 equations are removed.

We report the degree of regularity $D_{\mathrm{reg}}^{\mathrm{Exp}}$ reached during a Gröbner basis computation of a system of $m = n - \Delta$ equations in $n - \Delta$ variables coming from a HFE public-key generated from a univariate polynomial in $\mathbb{F}_{2^n}[X]$ of degree D. We also reported the degree of regularity $D_{\mathrm{reg}}^{\mathrm{Theo}}$ of a semi-regular system of the same size (as in Table (10)).

The experimental results on HFE-, no vinegar, are not completely conclusive. Whilst the degree of regularity appears to increase, it seems difficult to predict its behavior in function of the number of equations removed. This was also observed in [62] where the authors advised against using the minus modifier alone. Thus, the minus modifier should not be used alone.

We now consider the opposite situation, i.e. no minus and we increase the number of vinegar variables, i.e. HFEv.

n	Δ	$n-\Delta$	D	$D_{\mathrm{reg}}^{\mathrm{Theo}}$	$D_{\mathrm{reg}}^{\mathrm{Exp}}$
32	0	32	4	7	3
33	1	32	4	7	3
34	2	32	4	7	3
35	**3**	32	4	7	**4**
36	4	32	4	7	4
37	5	32	4	7	4
38	6	32	4	7	4
39	7	32	4	7	4
40	**8**	32	4	7	**5**
41	9	32	4	7	5
42	10	32	4	7	5
43	11	32	4	7	5
44	12	32	4	7	5
45	13	32	4	7	5
46	**14**	32	4	7	**6**
47	15	32	4	7	6
48	16	32	4	7	6
49	17	32	4	7	6
49	18	32	4	7	6
50	19	32	4	7	6
51	20	32	4	7	6

n	Δ	$n-\Delta$	D	$D_{\mathrm{reg}}^{\mathrm{Theo}}$	$D_{\mathrm{reg}}^{\mathrm{Exp}}$
41	0	41	4	8	3
42	1	41	4	8	3
43	2	41	4	8	3
44	**3**	41	4	8	**4**
45	4	41	4	8	4
46	5	41	4	8	4
47	6	41	4	8	4
48	7	41	4	8	4

Table 13: HFE- with $D = 4$; 32 and 41 equations.

n	Δ	$n-\Delta$	D	$D_{\mathrm{reg}}^{\mathrm{Theo}}$	$D_{\mathrm{reg}}^{\mathrm{Exp}}$
32	0	32	17	7	4
33	1	32	17	7	4
34	2	32	17	7	4
35	**3**	32	17	7	**5**
36	4	32	17	7	5
37	**5**	32	17	7	**6**
38	6	32	17	7	6
39	7	32	17	7	6

n	Δ	$n-\Delta$	D	$D_{\mathrm{reg}}^{\mathrm{Theo}}$	$D_{\mathrm{reg}}^{\mathrm{Exp}}$
41	0	41	17	8	4
42	1	41	17	8	4
43	2	41	17	8	4
44	**3**	41	17	8	**5**
45	4	41	17	8	5

Table 14: HFE- with $D = 17$; 32 and 41 equations.

n	v	$m = n - v$	D	$D_{\mathrm{reg}}^{\mathrm{Theo}}$	$D_{\mathrm{reg}}^{\mathrm{Exp}}$
32	**0**	32	6	7	**3**
32	**7**	25	6	7	**5**
32	**8**	25	6	7	**6**
32	9	25	6	7	6
32	10	25	7	7	6
32	**11**	25	6	7	**7**
32	12	25	6	7	7
32	15	25	6	7	7

Table 15: HFEv, $D = 6$ and 32 variables.

The experimental results are more stable. In all cases, we need to add 3 vinegar variables to increase the degree of regularity by 1.

n	v	$m = n - v$	D	$D_{\text{reg}}^{\text{Theo}}$	$D_{\text{reg}}^{\text{Exp}}$
25	**0**	25	9	6	**3**
26	**1**	25	9	6	**4**
27	2	25	9	6	4
28	3	25	9	6	4
29	**4**	25	9	6	**5**
30	5	25	9	6	5
31	6	25	9	6	5
32	7	25	9	6	**6**

Table 16: `HFEv`, $D = 9$ and 25 variables.

n	v	$m = n - v$	D	$D_{\text{reg}}^{\text{Theo}}$	$D_{\text{reg}}^{\text{Exp}}$
25	**0**	25	16	6	**3**
26	**1**	25	16	6	**4**
27	2	25	16	6	4
28	3	25	16	6	4
29	**4**	25	16	6	**5**
30	5	25	16	6	5
31	6	25	16	6	5
32	**7**	25	16	6	**6**

n	v	$m = n - v$	D	$D_{\text{reg}}^{\text{Theo}}$	$D_{\text{reg}}^{\text{Exp}}$
32	**0**	32	16	7	**3**
33	**1**	32	16	7	**4**
34	2	32	16	7	4
35	3	32	16	7	4
36	**4**	32	16	7	**5**
37	5	32	16	7	5

Table 17: `HFEv` with $D = 16$; 25 and 32 equations.

We also performed experimental results with a combination of vinegar and minus. Similarly to [62], we observed that the behaviour obtained seems similar for `HFEv-` with $\Delta = 0$ and v vinegar variables than for a `HFEv-` with $\Delta = v/2$ and $v/2$ vinegar variables.

8.3.4 Distinguishing-based attack against `HFEv-`

The idea of the so-called hybrid attack discussed in Section 8.3.1 is to combine exhaustise search with Gröbner bases. In [26], the authors propose an improved version of this hybrid attack that takes into account the specific structure of a `HFEv-` public system.

From (14), we can observe that the degree of regularity increases linearly with the number of minus or vinegar variables but logarithmically in the degree D. The strategy of [26] is to turn this remark into a distinguisher. Vinegar variables have an impact on the degree of regularity and so on the cost of a Gröbner basis computation.

More precisely, this attack reduces a `HFEv-` system to a `HFE-` system, by removing the vinegar variables one by one. To do so, k linear equations are added to the key-recovery system (7). We obtain a projected system $\mathbf{p}'$. If a linear combination of these k equations is equivalent to remove one vinegar variable, $\mathbf{p}'$ will be easier to solve with a Gröbner basis algorithm. In particular, the degree of regularity will decrease. This permits to detect the case where the k equations indeed eliminate a vinegar variable. Once these k equations found, the linear combination which removes one vinegar variable can be computed, then added to the initial system. The new system will be equivalent to the old system by removing one vinegar variable. By repeating this process, all vinegar variables can be eliminated, and we obtain a `HFE-` system.

According to [26], the complexity of the distinguishing-based attack is

$$O\left(2^{n-k} \times 3\binom{n+v-k}{D_{\text{reg}}}^2\binom{n+v-k}{2}\right) \tag{15}$$

with a classical computer, and is

$$O\left(2^{\frac{n-k}{2}} \times 3\binom{n+v-k}{D_{\text{reg}}}^2\binom{n+v-k}{2}\right) \tag{16}$$

with a quantum computer.

However, the number of added equations k is upper bounded. Let $\bar{k}$ be this value, when at most $\bar{k}$ equations are added, the degree of regularity of a projected and unprojected system are the same (when these equations do not remove one vinegar variable). When at least $\bar{k}+1$ equations are added, the distinguishing based attack fails because the projected system cannot be distinguished anymore of a random system.

So, $\bar{k}$ is estimated as following. We estimate d the degree of regularity of the projected system with Equation (14). Then, we estimate the degree of regularity of a random system with m equations and n' variables with the smallest index i such as the term z^i of G (Equation (17)) is zero or negative.

$$G(z) = \frac{(1+z)^{n'}}{(1+z^2)^m}. \tag{17}$$

We obtain $\bar{k}$ by searching the larger value k such as d is less or equal to the degree of regularity of a random system with $n' = n+v-k$ variables and $m = n-\Delta$ equations. When k equations are added, k variables are removed.

In Table 18, we take the minimum values of m and D for each level of security of `HFEv-`, and for $\Delta = v$, we give the values of v which permits to achieve the security level against the distinguishing based attack. We selected all our parameters taking into account the distinguishing-based attack.

(λ, m, D)	D_{reg} (14)	$\bar{k}$	Distinguishing based attack (15)
$(128, 162, 17)$	7	102	$v \geqslant 4$
$(192, 243, 17)$	10	144	$v \geqslant 8$
$(256, 324, 17)$	12	193	$v \geqslant 11$

Table 18: Values of v which reaches the security level against the distinguishing-based attack.

8.4 Key-recovery attacks

We conclude this part by covering key-recovery attacks. This part discusses the so-called *Kipnis-Shamir attack* [50] (Section 8.4.1) and differential attacks (Section 8.4.3).

8.4.1 Kipnis-Shamir attack

In [50], A. Kipnis and A. Shamir demonstrated that key-recovery in HFE is essentially equivalent to the problem of finding a low-rank linear combination of a set of m boolean matrices of size $m \times m$. This is a particular instance of the MinRank problem [18, 21].

We briefly review the principle of this attack for HFE. In the context of this attack, we can assume w.l.o.g. that the HFE polynomial has a simpler form:

$$\sum_{\substack{0 \leqslant j < i < n \\ 2^i + 2^j \leqslant D}} A_{i,j}\, X^{2^i + 2^j} \in \mathbb{F}_{2^n}[X], \text{ with } A_{i,j} \in \mathbb{F}_{2^n}. \tag{18}$$

We can then write (18) in a matrix form, that is:

$$\underline{X}\mathbf{F}\underline{X}^{\mathrm{T}}$$

with $\underline{X} = (X, X^2, X^{2^2}, \ldots, X^{2^{n-1}})$ and $\mathbf{F} \in \mathcal{M}(\mathbb{F}_{2^n})_{n \times n}$ is a symmetric matrix with zeroes on the diagonal (i.e. skew-symmetric matrix). Since the degree of F is bounded by D, it is easy to see that $\mathbf{F}$ has rank at most $\lceil \log_2(D) \rceil$. This implies that there exists a linear combinations of rank $\lceil \log_2(D) \rceil$ of the public matrices representing the public quadratic forms [9]. The secret-key can be then recovered easily from a solution of MinRank [50, 9].

In [9], the authors evaluated the cost of the Kipnis-Shamir key-recovery attack with the best known tools for solving the MinRank [36] instance that occurs in HFE. Following [9], the cost of the Kipnis-Shamir attack against HFE can be estimated to:

$$O\big(n^{\omega(\lceil \log_2(D) \rceil + 1)}\big), \text{ with } 2 \leq \omega \leq 3 \text{ being the linear algebra constant}$$

and where D is the degree of the secret univariate polynomial.

Until recently, it was not clear how to apply the key-recovery attack from [50, 9] to HFE- when $n - m \geq 2$. In [65], the authors explained how to extend MinRank-based key-recovery for all parameters of HFE-. Their results can be summarized as follows. From key-recovery point of view, HFE- with a secret univariate polynomial of degree D and n variables is equivalent to a HFE with m variables with secret univariate polynomial of degree $D \times 2^{\Delta}$. Combining with [9], the cost of a MinRank-based key-recovery attack against HFE$-$ is then:

$$O\big(m^{\omega(\lceil \log_2(D) \rceil + \Delta + 1)}\big).$$

For MinRank-based key-recovery, the minus modifier has then a strong impact on the security.

In the case of HFEv, one can see that the rank of the corresponding matrix (see, for exemple [62]) will be increased by the number of vinegar variables. Combining with the previous result, the cost of solving MinRank in the case of HFEv- is then:

$$O\big(n^{\omega(\lceil \log_2(D) \rceil + v + \Delta + 1)}\big), \tag{19}$$

where D is the degree of the secret univariate polynomial.

For all the parameters proposed for scheme, assuming $\omega = 2$, the cost (19) is always much bigger than the cost of the best direct attack (Section 8.1).

8.4.2 MinRank attacks with projections

In Section 8.4, we only described the first step – the MinRank – of a Kipnis-Shamir key-recovery attack. Thus, the complexity 19 is a lower bound on the total cost of the Kipnis-Shamir key-recovery attack. In [26], the authors provide the cost of the second step, finding an equivalent secret-key, for such attack. According to [26], the cost of this second step is:

$$O\left(\binom{n+v+r}{\Delta+v+r}^2\binom{n-\Delta}{2}+(\Delta+v+r+1)^3\,2^{\Delta+r+1}\right). \tag{20}$$

with $r = \lceil \log_2(D) \rceil$.

The authors of [26] also propose a method to improve the MinRank step (Section 8.4). The idea is very similar to the one described in Section 8.3.4. We try to eliminate vinegar variables to decrease the degree of regularity with respect to a direct MinRank, and so the complexity (19). This attack, called project-then-MinRank attack, has complexity:

$$O\left(\binom{n+v+r-c}{\Delta+v+r-c}^2\binom{n-\Delta}{2}2^{c(r+\Delta+\sqrt{n-\Delta})-\binom{c+1}{2}}\right), 1 \leq c \leq v. \tag{21}$$

This is also a lower on the cost a full-recovery. Indeed, we also need to add the cost of (20).

In Table 19, we consider the parameters used for RedGeMSS. For such familly, the degree is the smallest ($D = 17$) and so the rank. Thanks to [26], we have now a rational to choose the number of vinegar variables. In particular, this leads to choose Δ and v to be equal.

Below, we computed the smallest values of v which permit to reach the three security levels in the case of RedGeMSS.

(λ, m, D)	project-then-MinRank (21)
$(128, 162, 17)$	$v \geqslant 3$
$(192, 243, 17)$	$v \geqslant 6$
$(256, 324, 17)$	$v \geqslant 8$

Table 19: Values of v which reaches the security level ($\Delta = v$).

8.4.3 Differential attack

We finally consider so-called *differential attacks*, introduced [29], are structural attacks that can be used to attack multivariate cryptosystems. Differential attacks turned to be very efficient, e.g. [29, 14] against SFLASH [58]; a popular multivariate-based signature based on the Matsumoto and Imai [52].

HFE is the successor, and a generalization, of [52]. Up to know, differential attacks have not really threatened the security of HFEv-. This is due to the fact the univariate polynomial used is much more complex than in [52] variants such as SFLASH [58]. In [20], the authors proved that variants of HFE, such as GeMSS, are immune against known differential attacks.

8.5 Deriving number of variables for GeMSS

At this stage, we have a methodology for fixing the minimal number of equations m (Table 8). We now need to derive the number of vinegar variables v and minus Δ required to achieve the degree of regularity corresponding to a given security level (Table 12). This is the most delicate point. According to the experiments performed in Section 8.3.3, and the insight provided by the key-recovery attacks (Section 8.4), we make the choice to balance v and Δ.

In addition, we need to fix the degree D of the HFEv polynomial. This will give the initial degree of regularity for a nude HFE (Table 11). For GeMSS, we consider a secret univariate polynomial of degree $D = 513$. This corresponds to a degree of regularity of 6 for a nude HFE, i.e. without any modifier. From our experiments, we consider that 3 modifiers allow to increase the degree of regularity by one. Idenpendently of this submission, the authors [61] also derived a similar rule; as one can see from (14).

In Table 20, we then derive the number of modifiers required as $v + \Delta = 3 \times \text{Gap}$, with Gap being the difference with the targeted degree of regularity minus the initial degree of regularity (6 here). We consider the number of equations m and the targeted degree of regularity as in Table 12. The third column of Table 20 gives the number of modifiers required. We present below the results for GeMSS(a similar analysis can be easily done for BlueGeMSS and RedGeMSS).

	m	D	Gap	$v+\Delta$
GeMSS128	162	513	$14-6=8$	24
GeMSS192	243	513	$20-6=14$	42
GeMSS256	324	513	$27-6=21$	63

Table 20: Numbers of modifiers required in GeMSS.

8.6 A general method to derive secure parameters

We are now in position to provide a general methodology to derive secure parameters for GeMSS. Following Section 8.3.1, the number of equations should be chosen such that:

$$m \geq 1.26 \cdot \lambda.$$

Thus, we can assume that $m = \alpha \cdot \lambda$ with $\alpha \geq 1.26$.

From (11), the degree of regularity D_{reg} required for a given security level should verify:

$$O\left(\binom{m}{D_{\text{reg}}}^2\right) \geq 2^{\lambda}.$$

Using a loose approximation of the binomial and ignoring the coefficient in the big-O, we get that:

$$D_{\text{reg}} \geq \frac{\lambda}{\log_2(m^2)} = \frac{\lambda}{2\log_2(\alpha \cdot \lambda)}.$$

The last step requires to compute the number of vinegar variables required to reach D_{reg}. We first need to have the initial degree of regularity. We can assume that this is a function of $\log_2(D)$;

as explained in Section 8.3.2. From table 11, we can interpolate an expression for the degree of regularity $D_{\text{reg}}^{\texttt{HFE}}$ of a nude HFE:

$$D_{\text{reg}}^{\texttt{HFE}} \approx 2.03 + 0.36 \log_2(D).$$

The number of modifiers, using the experimental rule of Section 8.5, can be then approximated by:

$$\Delta + v \approx \frac{3\lambda}{\log_2(m^2)} - 6.09 - 1.08 \log_2(D) = \frac{1.5\lambda}{\log_2(\alpha \cdot \lambda)} - 6.09 - 1.08 \log_2(D). \tag{22}$$

Below, we computed this approximation for the parameters of G*e*MSS.

(λ, m, D)	Approximation (22) of $\Delta + v$
$(128, 162, 513)$	10.35
$(192, 243, 513)$	20.53
$(256, 324, 513)$	30.23

This has to be compared with the exact values provided in Table 20. The difference is mainly due to the loose approximation of the binomial for deriving (22). However, we can see that (22) captures the global trend and can be used to derive others secure parameters.

We can see that there is two strategies to derive secure parameters. In G*e*MSS, the goal is to minimize the size of the public-key. To do so, we are taking $m = 1.26 \cdot \lambda$. From (22), we can see that the number of modifiers decreases when D increases. We take the same number of vinegar variables v and the same number of minus Δ. To minimize the total number of variables m, we have then to increase the degree D of the univariate polynomial. However, the time to sign increases with D.

The strategy differs if the goal is to have a faster signing process together with a shorter signature. In this case, we have to take m bigger than $1.26 \cdot \lambda$. As a consequence, the number of iterations nb_ite can be decreased. We repeat then less the inversion process G*e*MSS.$\text{Inv}_{\mathbf{p}}$ in the signing process (Algorithm 4). The verification will be also faster. From (22), we can see that maximizing the number of modifiers makes possible to choose smaller D. However, this will increase the number of vinegar variables v and so the total number of variables m.

9 A larger family of G*e*MSS parameters

In [55], NIST announced the second round candidates and also provided some recommendations for the selected candidates. The goal of this part is to address the comments from [55] regarding G*e*MSS. The parameters proposed for G*e*MSS in the first round were very conservative in term of security. [55] suggests to explore different parameters in order to improve efficiency. We address this comment as follows.

- In Section 9.6, we present an exhaustive table including possible parameters and the corresponding timings.
- In Section 9.5, we explore the use of sparse polynomials in G*e*MSS to improve the efficiency of the signing process.

- We then suggest 3 sets of parameters for each security level with several trade-offs. This includes the initial parameters of GeMSS proposed in the first round, and two new more aggressive parameters (BlueGeMSS and RedGeMSS).
- We design a family of possible values that depends on only one parameter n. We call this family FGeMSS(n).

9.1 Set 1 of parameters: GeMSS (see Section 3)

The first set, that we GeMSS family, was the parameters proposed for the first round.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	equations	variables	$\lvert pk \rvert$ (KB)	$\lvert sk \rvert$ (bits)	sign (bits)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	162	186	352.19	128	258
GeMSS192	$(192, 513, 265, 22, 20, 4)$	243	285	1237.96	192	411
GeMSS256	$(256, 513, 354, 30, 33, 4)$	324	387	3040.70	256	576

Table 21: Summary of the parameters of GeMSS.

9.2 Set 2 of parameters: RedGeMSS

We call RedGeMSS the schemes described in Table 22. The public-key of RedGeMSS128 is 1.065 times larger than GeMSS128, the time to sign with RedGeMSS128 is 228 times faster than GeMSS128. This is because we use a smaller D.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	equations	variables	$\lvert pk \rvert$ (KB)	$\lvert sk \rvert$ (bits)	sign (bits)
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	162	192	375.21	128	282
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	243	291	1290.54	192	435
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	324	393	3135.59	256	600

Table 22: Summary of the parameters of RedGeMSS.

9.3 Set 3 of parameters: BlueGeMSS

We call BlueGeMSS the schemes described in Table 23. The public-key of BlueGeMSS128 is 1.032 times larger than GeMSS128, the time to sign with BlueGeMSS128 is 6.53 times faster than GeMSS128. This is because we use a smaller D.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	equations	variables	$\|pk\|$ (KB)	$\|sk\|$ (bits)	sign (bits)
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	162	189	363.61	128	270
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	243	288	1264.12	192	423
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	324	390	3087.96	256	588

Table 23: Summary of the parameters of BlueGeMSS.

9.4 FGeMSS(n) family

In multivariate schemes, we have many parameters that can be adjusted. This is an advantage since, for example, for a given security we can decrease the time to sign if we increase the length of the public-key, i.e. some interesting tradeoffs are possible. However, when a new cryptanalysis idea is found, it is not always easy for a non multivariate specialist to see how to adjust the parameters in order to maintain a given security level against the best known attacks. For example, when RSA-512 was factored, it was natural to suggest to use a larger modulo and to look at what value of n should be used from the best known attacks (instead of designing another scheme). But when an attack on `QUARTZ` was published with a security expected [38] to be slightly smaller than 2^{80} it was not so easy to adjust the security parameters since we have here many possibilities. Therefore, we see that it is sometime convenient to have a "*dimension 1*" family instead of a single point (like `QUARTZ`) or a many dimension family (like the variants of `HFE`).

We present here such "*dimension 1*" family, called FGeMSS(n). It is such that:

- $\text{nb\_ite} = 1$
- n is again $m + \Delta$
- $\Delta + v = 21 + \lceil 0.11(n - 266) \rceil, \Delta = \left\lfloor \frac{\Delta+v}{2} \right\rfloor$ and $v = \left\lceil \frac{\Delta+v}{2} \right\rceil$
- D is the maximum sum of two power of two smaller or equal to $129 + \lceil 4.2(n - 266) \rceil$.

The public-key is a system in $\mathbb{F}_2$ with $n - \Delta$ equations and $n + v$ variables.

For example, we obtain the following parameters.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	equations	variables	$\|pk\|$ (KB)	$\|sk\|$ (bits)	sign (bits)
FGeMSS(266)	$(128, 129, 266, 10, 11, 1)$	256	277	1232.13	128	277
FGeMSS(402)	$(192, 640, 402, 18, 18, 1)$	384	420	4243.73	192	420
FGeMSS(537)	$(256, 1152, 537, 25, 26, 1)$	512	563	10161.09	256	563

Table 24: Parameters of FGeMSS.

It can be emphasized that FGeMSS can be nicely combined with DualModeMS [39]. DualModeMS is a generic technique permitting to transform any Matsumoto-Imai based multivariate signature scheme into a new scheme with much shorter public-key but larger signatures. In the case of FGeMSS266, we will typically get a public-key of 512 bytes with a signature size of about 32 KB.

9.5 SparseGeMSS

In this section, we introduce s, a new security parameter. We propose to remove s terms in the `HFEv` polynomial to improve the efficiency of the signing process. When s is small, we think the security is not impacted by this change, whereas we can obtain a factor at most two for the signing process. This method is new and so a new analysis of security is required.

The improvement is based on the fact that during the computation of the Frobenius map, a $(2D-2)$-degree square in $\mathbb{F}_{2^n}$ is computed, then is reduced modulo F. In binary fields, all odd degree terms of a square are null, because of the linearity of the Frobenius endomorphism. Then, we remark that the Euclidean division of B a square by a square implies that the quotient Q is a square. F is not a square because it contains the terms X^{2^0} and X^{2^i+1} for $0 < i \leq \lfloor \log_2(D) \rfloor$. However, the gap between the odd degrees 2^j+1 and $2^{j+1}+1$ is 2^j. This gap increases fastly when j increases. So, if we take $D = 2^k+2$, then we remove the s largest odd degrees ($s \leq k$), we obtain a `HFE` polynomial $F = F_0 + X^{2^{k-s}+2} F_1$ with F_0 a $(2^{k-s}+1)$-degree polynomial and F_1 a $(2^k - 2^{k-s})$-degree square. By removing only one term ($s = 1$), the high half of F is square.

Now, we exploit the fact that F_1 is a square. This implies $Q = Q_0 + X^{2^{k-s}} Q_1$ with Q_0 a $(2^{k-s}-1)$-degree polynomial and Q_1 a $(2^k - 2^{k-s})$-degree square. Moreover, the classical Euclidean division algorithm is equivalent to compute the product of Q by F, then to add it to B. So, if Q_1 is a square, we avoid the half of the multiplications for this part of Q. The size of Q_1 is $(2^k - 2^{k-s}+1)$, so we avoid $2^{k-1} - \lfloor 2^{k-s-1} \rfloor$ multiplications in $\mathbb{F}_{2^n}$.

When $s = k$, Q is a square and the speed-up is maximal. It is about $\frac{2^k+1}{2^{k-1}+1} < 2$. When $s = k+1$, F, Q and the remainder are squares. However, this value of s decreases the security. The D-degree `HFE` polynomial F is equivalent to a $\frac{D}{2}$-degree `HFE` polynomial (by taking $Y = X^2$), so the degree of regularity depends on $\frac{D}{2}$. In this case, D could be multiplied by two, but this would remove the factor 2 obtained with our strategy.

Degree of regularity. We have measured the $D_{\text{reg}}^{\text{Exp}}$ observed in practice for `HFE` in function of s. The results are summarized in Table 25. When s is small, the degree of regularity is not impacted. For the largest value of s, the degree of regularity decrements. As soon as D is multiplied by two, we have observed that the degree of regularity does not decrement anymore.

MinRank. The security of `HFE` against the Kipnis-Shamir attacks (Section 8.4.1) seems not to be impacted by the parameter s. This implies to vanish the s last coefficients in the first column of $\mathbf{F}$. However, the first coefficient of $\mathbf{F}$ corresponds to X^2 which has an even degree, so the rank does not decrease. We remark also that the last row of $\mathbf{F}$ is not null, since the monic coefficient corresponding to X^{2^k+2} is present.

SparseGeMSS. With our trick, all previous families could become more efficient by using their "sparse" version. To do this transformation, we increment D (when D is odd) and we set $s = 3$. In this way, we avoid 43.75% of the multiplications (when D is odd) in the modular reduction by F. When $D \neq (2^{\lfloor \log_2(D) \rfloor}+2)$ is even, the speed-up is different because our trick improves the modular reduction when $s = 0$ (because $Q = Q_0 + X^{2^{\lfloor \log_2(D) \rfloor}+2} Q_1$ with Q_1 a $(D - 2^{\lfloor \log_2(D) \rfloor} - 2)$-degree square, so we avoid $\frac{D-2^{\lfloor \log_2(D) \rfloor}-2}{2} \neq 0$ multiplications in $\mathbb{F}_{2^n}$). We take a small value of s to be

Minimal m	HFE(D)	s	$D_{\text{reg}}^{\text{Exp}}$
$\geqslant 9$	17	0	4
$\geqslant 15$	18	$s \leqslant 3$	4
$160 \geqslant m \geqslant 5$		$4 \leqslant s \leqslant 5$	3
$\geqslant 16$	129	0	5
$\geqslant 16$	130	$s \leqslant 5$	5
$\geqslant 18$		6	5
$\geqslant 23$		7	5
$70 \geqslant m \geqslant 9$		8	4
$\geqslant 24$	513	0	6
$\geqslant 24$	514	$s \leqslant 6$	6
$\geqslant 25$		7	6
$35 \geqslant m \geqslant 16$		$8 \leqslant s \leqslant 10$	5
$\geqslant 32$	4097	0	7
$\geqslant 32$	4098	$s \leqslant 10$	7
$\geqslant 33$		11	7
$35 \geqslant m \geqslant 24$		$12 \leqslant s \leqslant 13$	6

Table 25: Degree of regularity in the case of `HFE` algebraic systems, in function of s. The maximum value of s is $\lfloor \log_2(D) \rfloor + 1$.

$$\mathbf{F} = \begin{pmatrix} * & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 \\ \mathbf{0} & * & * & 0 & 0 \\ \mathbf{0} & * & * & * & 0 \\ \mathbf{0} & 1 & 0 & 0 & 0 \end{pmatrix}$$

Figure 2: Example of matrix $\mathbf{F} \in \mathcal{M}(\mathbb{F}_{2^n})$ for $D = 18$ and $s = 3$. The three removed coefficients are in bold. Since the coefficients are in a binary field, the matrix is not symmetric.

secure, but enough large to obtain an interesting speed-up. The Frobenius map is the core of the signing process, so this factor remains approximately the same for the signing process. However, this method is not interesting for small degrees, because the Frobenius map can be computed more fastly with multi-squaring tables (as in [62]). Experimentally, we keep the previous speed-up when $D \geq 514$, we lose a part when $D = 130$ and $n > 196$, and the method is completely useless when $D \leq 34$. For this reason, we give the possibility to use SparseGeMSS only for the degrees D strictly greater than 127.

9.6 An exhaustive table for the choice of the parameters

We propose here a large number of security parameters. For different values of D and for nb_ite from 1 to 4, we take the smallest m such that $(m, \text{nb\_ite})$ respects Theorem 2. Then, we deduce the number of modifiers, and so Δ and v. Finally, when $D > 127$, we take $s = 0$ then $s = 3$ (as described in Section 9.5). In Table 26, we give the performance of these parameters with our best version of `MQsoft` [40, 1].

For nb_ite < 3, the number of equations is a multiple of 8. So, the public-key is naturally stored with the packed representation (Section 2.6.5). This implies the theoretical size of the public-key is reached without to decrease performances. For the other values of m, the performance of the verifying process decreases when m mod 8 increases.

$(\lambda, D, n, \Delta, v, \text{nb\_ite}, s)$	key gen. (MC)	sign (MC)	verify (KC)	$\|pk\|$ (KB)	$\|sk\|$ (B)	sign (bits)
$(128, 17, 268, 12, 12, 1, 0)$	152	2.27	37.8	1260	16	280
$(128, 17, 204, 12, 15, 2, 0)$	61.3	2.17	54	578	16	246
$(128, 17, 186, 15, 15, 3, 0)$	45.5	2.01	94.7	434	16	261
$(\mathbf{128}, \mathbf{17}, \mathbf{177}, \mathbf{15}, \mathbf{15}, \mathbf{4}, \mathbf{0})$	39.5	2.33	141	375	16	282
$(128, 33, 268, 12, 12, 1, 0)$	155	6.43	38.4	1260	16	280
$(128, 33, 204, 12, 15, 2, 0)$	62.2	6.31	54	578	16	246
$(128, 33, 186, 15, 15, 3, 0)$	46.6	5.74	94.9	434	16	261
$(128, 33, 177, 15, 15, 4, 0)$	40.1	7.05	142	375	16	282
$(128, 129, 266, 10, 11, 1, 0)$	155	62.1	37.4	1230	16	277
$(128, 130, 266, 10, 11, 1, 3)$	155	40.3	38.1	1230	16	277
$(128, 129, 204, 12, 12, 2, 0)$	62.6	62.5	53.5	562	16	240
$(128, 130, 204, 12, 12, 2, 3)$	62.4	42.2	51.9	562	16	240
$(128, 129, 185, 14, 13, 3, 0)$	45.4	66.1	107	421	16	252
$(128, 130, 185, 14, 13, 3, 3)$	45	37.8	106	421	16	252
$(\mathbf{128}, \mathbf{129}, \mathbf{175}, \mathbf{13}, \mathbf{14}, \mathbf{4}, \mathbf{0})$	39.2	81.3	136	364	16	270
$(128, 130, 175, 13, 14, 4, 3)$	39.2	47	136	364	16	270
$(128, 513, 265, 9, 9, 1, 0)$	157	466	40.9	1210	16	274
$(128, 514, 265, 9, 9, 1, 3)$	156	258	37.7	1210	16	274
$(128, 513, 202, 10, 11, 2, 0)$	62.1	459	50	547	16	234
$(128, 514, 202, 10, 11, 2, 3)$	61.6	271	51.8	547	16	234
$(128, 513, 183, 12, 12, 3, 0)$	44.6	413	104	408	16	243
$(128, 514, 183, 12, 12, 3, 3)$	44.4	244	103	408	16	243
$(\mathbf{128}, \mathbf{513}, \mathbf{174}, \mathbf{12}, \mathbf{12}, \mathbf{4}, \mathbf{0})$	38.7	531	106	352	16	258
$(128, 514, 174, 12, 12, 4, 3)$	38.7	331	106	352	16	258

$(\lambda, D, n, \Delta, v, \text{nb\_ite}, s)$	key gen. (MC)	sign (MC)	verify (KC)	$\|pk\|$ (KB)	$\|sk\|$ (B)	sign (bits)
$(192, 17, 404, 20, 19, 1, 0)$	807	5.87	125	4300	24	423
$(192, 17, 310, 22, 23, 2, 0)$	267	4.52	159	2000	24	378
$(192, 17, 279, 23, 25, 3, 0)$	192	5.06	199	1480	24	400
$(\mathbf{192}, \mathbf{17}, \mathbf{266}, \mathbf{23}, \mathbf{25}, \mathbf{4}, \mathbf{0})$	173	5.97	334	1290	24	435
$(192, 33, 404, 20, 19, 1, 0)$	813	16	124	4300	24	423
$(192, 33, 310, 22, 23, 2, 0)$	270	13	160	2000	24	378
$(192, 33, 279, 23, 25, 3, 0)$	197	17.5	201	1480	24	400
$(192, 33, 266, 23, 25, 4, 0)$	175	22.3	337	1290	24	435
$(192, 129, 402, 18, 18, 1, 0)$	808	145	123	4240	24	420
$(192, 130, 402, 18, 18, 1, 3)$	811	108	124	4240	24	420
$(192, 640, 402, 18, 18, 1, 0)$	833	1580	123	4240	24	420
$(192, 640, 402, 18, 18, 1, 3)$	829	964	125	4240	24	420
$(192, 129, 308, 20, 22, 2, 0)$	272	129	157	1970	24	372
$(192, 130, 308, 20, 22, 2, 3)$	271	84.7	159	1970	24	372
$(192, 129, 278, 22, 23, 3, 0)$	196	198	196	1450	24	391
$(192, 130, 278, 22, 23, 3, 3)$	197	136	191	1450	24	391
$(\mathbf{192}, \mathbf{129}, \mathbf{265}, \mathbf{22}, \mathbf{23}, \mathbf{4}, \mathbf{0})$	174	252	325	1260	24	423
$(192, 130, 265, 22, 23, 4, 3)$	174	162	323	1260	24	423
$(192, 513, 399, 15, 18, 1, 0)$	812	1110	121	4180	24	417
$(192, 514, 399, 15, 18, 1, 3)$	819	715	122	4180	24	417
$(192, 513, 308, 20, 19, 2, 0)$	273	943	154	1930	24	366
$(192, 514, 308, 20, 19, 2, 3)$	271	540	156	1930	24	366
$(192, 513, 276, 20, 22, 3, 0)$	198	1450	189	1430	24	382
$(192, 514, 276, 20, 22, 3, 3)$	197	824	189	1430	24	382
$(\mathbf{192}, \mathbf{513}, \mathbf{265}, \mathbf{22}, \mathbf{20}, \mathbf{4}, \mathbf{0})$	175	1800	304	1240	24	411
$(192, 514, 265, 22, 20, 4, 3)$	174	1050	305	1240	24	411

$(\lambda, D, n, \Delta, v, \text{nb\_ite}, s)$	key gen. (MC)	sign (MC)	verify (KC)	$\|pk\|$ (KB)	$\|sk\|$ (B)	sign (bits)
$(256, 17, 540, 28, 29, 1, 0)$	2840	11.5	380	10400	32	569
$(256, 17, 415, 31, 32, 2, 0)$	968	8.6	387	4810	32	510
$(256, 17, 375, 33, 33, 3, 0)$	611	8.17	610	3570	32	540
$(\mathbf{256}, \mathbf{17}, \mathbf{358}, \mathbf{34}, \mathbf{35}, \mathbf{4}, \mathbf{0})$	534	9.82	704	3140	32	600
$(256, 33, 540, 28, 29, 1, 0)$	2860	31.2	381	10400	32	569
$(256, 33, 415, 31, 32, 2, 0)$	978	28	385	4810	32	510
$(256, 33, 375, 33, 33, 3, 0)$	611	28.2	624	3570	32	540
$(256, 33, 358, 34, 35, 4, 0)$	527	35.3	722	3140	32	600
$(256, 129, 540, 28, 26, 1, 0)$	2880	313	374	10300	32	566
$(256, 130, 540, 28, 26, 1, 3)$	2880	226	369	10300	32	566
$(256, 129, 414, 30, 30, 2, 0)$	973	302	375	4740	32	504
$(256, 130, 414, 30, 30, 2, 3)$	975	222	363	4740	32	504
$(256, 129, 372, 30, 33, 3, 0)$	606	328	582	3510	32	531
$(256, 130, 372, 30, 33, 3, 3)$	604	207	606	3510	32	531
$(\mathbf{256}, \mathbf{129}, \mathbf{358}, \mathbf{34}, \mathbf{32}, \mathbf{4}, \mathbf{0})$	530	399	684	3090	32	588
$(256, 130, 358, 34, 32, 4, 3)$	530	264	689	3090	32	588
$(256, 513, 537, 25, 26, 1, 0)$	2900	2510	372	10200	32	563
$(256, 514, 537, 25, 26, 1, 3)$	2900	1430	367	10200	32	563
$(256, 1152, 537, 25, 26, 1, 0)$	2920	7150	356	10200	32	563
$(256, 1152, 537, 25, 26, 1, 3)$	2920	4510	368	10200	32	563
$(256, 513, 414, 30, 27, 2, 0)$	974	2430	356	4680	32	498
$(256, 514, 414, 30, 27, 2, 3)$	976	1360	361	4680	32	498
$(256, 513, 372, 30, 30, 3, 0)$	611	2240	554	3460	32	522
$(256, 514, 372, 30, 30, 3, 3)$	609	1370	565	3460	32	522
$(\mathbf{256}, \mathbf{513}, \mathbf{354}, \mathbf{30}, \mathbf{33}, \mathbf{4}, \mathbf{0})$	530	3020	678	3040	32	576
$(256, 514, 354, 30, 33, 4, 3)$	527	1690	669	3040	32	576

Table 26: Performance of an exhaustive set of security parameters. We use a Skylake processor (LaptopS). The results have three significant digits. The parameters in bold correspond to RedG*e*MSS, BlueG*e*MSS and G*e*MSS.

10 Advantages and limitations (2.B.6)

Since the first scheme of Matsumoto and Imai [52] in 1988, almost 30 years ago, multivariate-based cryptosystems have been extensively analysed in the literature. We have designed G*e*MSS using this knowledge and derive a general methodology to derive parameters. We then proposed three set of parameters: G*e*MSS, the more conservative, and BlueRed/RedG*e*MSS that are more efficient (but also, more agressive in term of security). We also performed practical experiments using the best known tools for computing Gröbner bases.

From a practical point of view, the main drawback of G*e*MSS is the size of the public-key. However, we mention that the generation of a (public-key, secret-key) remains rather efficient in G*e*MSS. The main advantages of G*e*MSS are the size of the signatures generated, about 2λ bits, and the fast verification process.

References

[1] MQsoft: a fast multivariate cryptography library, December 2018. https://www-polsys.lip6.fr/Links/NIST/MQsoft.html.

[2] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison between XL and gröbner basis algorithms. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 2004.

[3] Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, Université de Paris VI, 2004.

[4] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *International Conference on Polynomial System Solving – ICPSS*, pages 71–75, 2004.

[5] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the Complexity of the F5 Gröbner basis Algorithm. *Journal of Symbolic Computation*, pages 1–24, September 2014. 24 pages.

[6] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Bo-Yin Yang. Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In *The Effective Methods in Algebraic Geometry Conference – MEGA 2005*, pages 1–14, 2005.

[7] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic boolean systems. *Journal of Complexity*, 29(1):53–75, February 2013.

[8] Daniel J. Bernstein and Bo-Yin Yang. Asymptotically faster quantum algorithms to solve multivariate quadratic equations. In *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, pages 487–506, 2018.

[9] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Cryptanalysis of hfe, multi-hfe and variants for odd and even characteristic. *Des. Codes Cryptography*, 69(1):1–52, 2013.

[10] Olivier Billet and Jintai Ding. *Overview of Cryptanalysis Techniques in Multivariate Public Key Cryptography*, pages 263–283. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[11] Andrey Bogdanov, Thomas Eisenbarth, Andy Rupp, and Christopher Wolf. Time-area optimized public-key engines: Mq-cryptosystems as replacement for elliptic curves? *IACR Cryptology ePrint Archive*, 2008:349, 2008.

[12] Wieb Bosma, John J. Cannon, and Catherine Playoust. The Magma algebra system I: The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[13] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in F_2. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2010.

[14] Charles Bouillaguet, Pierre-Alain Fouque, and Gilles Macario-Rat. Practical key-recovery for all possible parameters of SFLASH. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 667–685. Springer, 2011.

[15] Michael Brickenstein and Alexander Dreyer. Polybori: A framework for gröbner-basis computations with boolean polynomials. *J. Symb. Comput.*, 44(9):1326–1345, 2009.

[16] Bruno Buchberger. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation*, 41(3-4):475–511, 2006.

[17] Bruno Buchberger, Georges E. Collins, Rudiger G. K. Loos, and Rudolph Albrecht. Computer algebra symbolic and algebraic computation. *SIGSAM Bull.*, 16(4):5–5, 1982.

[18] Jonathan F Buss, Gudmund S Frandsen, and Jeffrey O Shallit. The computational complexity of some problems of linear algebra. *Journal of Computer and System Sciences*, 58(3):572–596, 1999.

[19] Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors. *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*. Springer, 2016.

[20] Ryann Cartor, Ryan Gipson, Daniel Smith-Tone, and Jeremy Vates. On the differential security of the hfev- signature primitive. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2016.

[21] Nicolas Courtois. Efficient zero-knowledge authentication based on a linear algebra problem minrank. In *ASIACRYPT*, volume 2248, pages 402–421. Springer, 2001.

[22] Nicolas Courtois. Generic attacks and the security of quartz. In *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 351–364. Springer, 2003.

[23] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

[24] Nicolas T. Courtois. Short signatures, provable security, generic attacks and computational security of multivariate polynomial schemes such as hfe, quartz and sflash. *IACR Cryptology ePrint Archive*, 2004:143, 2004.

[25] Jintai Ding and Thorsten Kleinjung. Degree of regularity for HFE-. *IACR Cryptology ePrint Archive*, 2011:570, 2011.

[26] Jintai Ding, Ray A. Perlner, Albrecht Petzoldt, and Daniel Smith-Tone. Improved cryptanalysis of hfev- via projection. In *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, pages 375–395, 2018.

[27] Jintai Ding and Bo-Yin Yang. *Multivariate Public Key Cryptography*, pages 193–241. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[28] Jintai Ding and Bo-Yin Yang. Degree of regularity for HFEv and HFEv-. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, volume 7932 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2013.

[29] Vivien Dubois, Pierre-Alain Fouque, Adi Shamir, and Jacques Stern. Practical cryptanalysis of SFLASH. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2007.

[30] Vivien Dubois and Nicolas Gama. The degree of regularity of HFE systems. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 557–576. Springer, 2010.

[31] Christian Eder and Jean-Charles Faugère. A survey on signature-based algorithms for computing gröbner bases. *J. Symb. Comput.*, 80:719–784, 2017.

[32] J.-C. Faugère. A new efficient algorithm for computing gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.

[33] J.-C. Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero : F5. In *ISSAC'02*, pages 75–83. ACM press, 2002.

[34] Jean-Charles Faugère. Algebraic cryptanalysis of HFE using Gröbner bases. Reasearch report RR-4738, INRIA, 2003.

[35] Jean-Charles Faugère. FGb: A Library for Computing Gröbner Bases. In Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 84–87, Berlin, Heidelberg, September 2010. Springer Berlin / Heidelberg.

[36] Jean-Charles Faugère, Mohab Safey El Din, and Pierre-Jean Spaenlehauer. On the complexity of the generalized minrank problem. *Journal of Symbolic Computation*, 55:30–58, 2013.

[37] Jean-Charles Faugère, Kelsey Horan, Delaram Kahrobaei, Marc Kaplan, Elham Kashefi, and Ludovic Perret. Fast quantum algorithm for solving multivariate quadratic equations. To appear.

[38] Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using gröbner bases. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2003.

[39] Jean-Charles Faugère, Ludovic Perret, and Jocelyn Ryckeghem. DualModeMS: A Dual Mode for Multivariate-based Signature. Research report, UPMC - Paris 6 Sorbonne Universités ; INRIA Paris ; CNRS, December 2017.

[40] Jean-Charles Faugère, Ludovic Perret, and Jocelyn Ryckeghem. Software toolkit for hfe-based multivariate schemes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):257–304, 2019.

[41] Pierre-Alain Fouque, Gilles Macario-Rat, Ludovic Perret, and Jacques Stern. Total break of the l-ic signature scheme. In Ronald Cramer, editor, *Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, Barcelona, Spain, March 9-12, 2008. Proceedings*, volume 4939 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2008.

[42] Giordano Fusco and Eric Bach. *Phase Transition of Multivariate Polynomial Systems*, pages 632–645. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[43] François Le Gall. Algebraic complexity theory and matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, page 23. ACM, 2014.

[44] Shuhong Gao, Yinhua Guan, and Frank Volny, IV. A new incremental algorithm for computing groebner bases. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 13–19, New York, NY, USA, 2010. ACM.

[45] Shuhong Gao, Frank Volny IV, and Mingsheng Wang. A new framework for computing gröbner bases. *Math. Comput.*, 85(297), 2016.

[46] Louis Goubin and Nicolas Courtois. Cryptanalysis of the TTM cryptosystem. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2000.

[47] Louis Granboulan, Antoine Joux, and Jacques Stern. Inverting HFE is quasipolynomial. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.

[48] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. ACM, 1996.

[49] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 1999.

[50] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[51] Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2190–2202. SIAM, 2017.

[52] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynominal-tuples for efficient signature-verification and message-encryption. In *EUROCRYPT*, volume 330 of *Lecture Notes in Computer Science*, pages 419–453. Springer, 1988.

[53] Mohamed Saied Emam Mohamed and Albrecht Petzoldt. The shortest signatures ever. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings*, volume 10095 of *Lecture Notes in Computer Science*, pages 61–77, 2016.

[54] NESSIE. New european schemes for signatures, integrity, and encryption, 2003.

[55] NIST. Status report on the first round of the nist post-quantum cryptography standardization process.

[56] Jacques Patarin. Cryptanalysis of the matsumoto and imai public key scheme of eurocrypt'88. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 1995.

[57] Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 1996.

[58] Jacques Patarin, Nicolas Courtois, and Louis Goubin. Flash, a fast multivariate signature algorithm. In *CT-RSA*, volume 2020 of *Lecture Notes in Computer Science*, pages 298–307. Springer, 2001.

[59] Jacques Patarin, Nicolas Courtois, and Louis Goubin. Quartz, 128-bit long digital signatures. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 282–297. Springer, 2001.

[60] Ludovic Perret. *Bases de Gröbner en Cryptographie Post-Quantique. (Gröbner bases techniques in Quantum-Safe Cryptography).* 2016.

[61] Albrecht Petzoldt. On the complexity of the hybrid approach on hfev-. Cryptology ePrint Archive, Report 2017/1135, 2017. https://eprint.iacr.org/2017/1135.

[62] Albrecht Petzoldt, Ming-Shing Chen, Bo-Yin Yang, Chengdong Tao, and Jintai Ding. Design principles for hfev- based multivariate signature schemes. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 311–334. Springer, 2015.

[63] ETSI ISG QSC. Quantum-safe cryptography (QSC); quantum-safe algorithmic framework. http://www.etsi.org/deliver/etsi_gr/QSC/001_099/001/01.01.01_60/gr_QSC001v010101p.pdf.

[64] Koichi Sakumoto, Taizo Shirai, and Harunaga Hiwatari. On provable security of UOV and HFE signature schemes against chosen-message attack. In *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 68–82. Springer, 2011.

[65] Jeremy Vates and Daniel Smith-Tone. Key recovery attack for all parameters of HFE-. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2017.

[66] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed)*. Cambridge University Press, 2013.

[67] Christopher Wolf. *Multivariate quadratic polynomials in public key cryptography*. Univ. Leuven Heverlee, 2005.

Appendix

A Space (April 1$^{\text{st}}$, 2019 version)

Here are the size of the public-key, secret-key and signature, as submitted at the beginning of the second round. The implementation did not optimize the size, so it explains the difference with theoretical sizes. Only the size of the signature was optimized.

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	$\|pk\|$ (KB)	$\|sk\|$ (KB)	sign (bits)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	352.188 / 417.408	13.43775 / 14.520	258 / 258
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	363.609 / 430.944	13.696375 / 14.664	270 / 270
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	375.21225 / 444.696	13.104 / 13.824	282 / 282
GeMSS192	$(192, 513, 265, 22, 20, 4)$	1237.9635 / 1304.192	34.069375 / 40.280	411 / 411
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	1264.116375 /1331.744	35.377375 / 41.720	423 / 423
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	1290.542625 / 1359.584	34.791125 / 40.760	435 / 435
GeMSS256	$(256, 513, 354, 30, 33, 4)$	3040.6995 / 3046.848	75.892125 / 83.688	576 / 576
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	3087.963 / 3094.200	71.4595 / 78.096	588 / 588
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	3135.591 / 3141.912	71.887375 / 78.408	600 / 600

Table 27: Memory cost, theoretical size / practical size. 1 KB is 1000 bytes.

B Time (April 1$^{\text{st}}$, 2019 version)

Here are the performance measurements of the initial version submitted for the second round.

B.1 Reference implementation

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (GC)	sign (MC)	verify (MC)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	1.88	6690	29.1
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	1.51	774	30
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	1.21	17.6	26.8
GeMSS192	$(192, 513, 265, 22, 20, 4)$	7.92	15100	89
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	6.72	1280	89
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	5.89	28	72.3
GeMSS256	$(256, 513, 354, 30, 33, 4)$	20.5	25300	172
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	19.4	1640	184
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	17.7	37.3	146

Table 28: Performance of the reference implementation. We use a Skylake processor (LaptopS). MC (resp. GC) stands for Mega (resp. Giga) Cycles. The results have three significant digits.

B.2 Optimized (Haswell) implementation

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	51.9	1220	150
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	52.9	202	158
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	55.3	5.57	162
GeMSS192	$(192, 513, 265, 22, 20, 4)$	273	3580	439
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	287	526	442
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	273	13.9	455
GeMSS256	$(256, 513, 354, 30, 33, 4)$	844	7090	943
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	874	1050	955
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	861	25.8	975

Table 29: Performance of the optimized implementation. We use a Haswell processor (ServerH). MC (resp. KC) stands for Mega (resp. Kilo) Cycles. The results have three significant digits.

B.3 Additional (Skylake) implementation

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	50.8	941	146
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	52.2	159	154
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	53	4.63	160
GeMSS192	$(192, 513, 265, 22, 20, 4)$	265	2890	436
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	266	430	441
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	266	11.8	453
GeMSS256	$(256, 513, 354, 30, 33, 4)$	872	4830	1020
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	889	691	1020
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	890	18.3	1050

Table 30: Performance of the additional implementation. We use a Skylake processor (LaptopS). MC (resp. KC) stands for Mega (resp. Kilo) Cycles. The results have three significant digits.

B.4 MQsoft

scheme	$(\lambda, D, n, \Delta, v, \text{nb\_ite})$	key gen. (MC)	sign (MC)	verify (KC)
GeMSS128	$(128, 513, 174, 12, 12, 4)$	38.5	750	82
BlueGeMSS128	$(128, 129, 175, 13, 14, 4)$	39.3	106	111
RedGeMSS128	$(128, 17, 177, 15, 15, 4)$	39.2	2.79	109
GeMSS192	$(192, 513, 265, 22, 20, 4)$	175	2320	239
BlueGeMSS192	$(192, 129, 265, 22, 23, 4)$	172	331	252
RedGeMSS192	$(192, 17, 266, 23, 25, 4)$	171	8.38	255
GeMSS256	$(256, 513, 354, 30, 33, 4)$	532	3640	566
BlueGeMSS256	$(256, 129, 358, 34, 32, 4)$	529	545	583
RedGeMSS256	$(256, 17, 358, 34, 35, 4)$	523	12.9	588

Table 31: Performance of `MQsoft`. We use a Skylake processor (LaptopS). MC (resp. KC) stands for Mega (resp. Kilo) Cycles. The results have three significant digits.

C An exhaustive table for the choice of the parameters (April 1st, 2019 version)

$(\lambda, D, n, \Delta, v, \text{nb\_ite}, s)$	key gen. (MC)	sign (MC)	verify (KC)	$\lvert pk \rvert$ (KB)	$\lvert sk \rvert$ (KB)	sign (bits)
$(128, 17, 268, 12, 12, 1, 0)$	153	2.19	36.6	1260	23.8	280
$(128, 17, 204, 12, 15, 2, 0)$	58	2.65	49.6	578	16.5	246
$(128, 17, 186, 15, 15, 3, 0)$	45.6	2.43	67.5	434	14.2	261
$(\mathbf{128}, \mathbf{17}, \mathbf{177}, \mathbf{15}, \mathbf{15}, \mathbf{4}, \mathbf{0})$	39.2	2.79	109	375	13.1	282
$(128, 33, 268, 12, 12, 1, 0)$	155	7.28	36.4	1260	24.4	280
$(128, 33, 204, 12, 15, 2, 0)$	58.4	8.54	50.5	578	17	246
$(128, 33, 186, 15, 15, 3, 0)$	45.8	7.68	66.3	434	14.7	261
$(128, 33, 177, 15, 15, 4, 0)$	39.8	8.82	111	375	13.5	282
$(128, 129, 266, 10, 11, 1, 0)$	154	82.5	36.2	1230	24.6	277
$(128, 130, 266, 10, 11, 1, 3)$	155	47	36.3	1230	24.5	277
$(128, 129, 204, 12, 12, 2, 0)$	59.2	101	48.6	562	16.2	240
$(128, 130, 204, 12, 12, 2, 3)$	59.2	61.5	49.2	562	16.2	240
$(128, 129, 185, 14, 13, 3, 0)$	44.9	84	68.7	421	14.4	252
$(128, 130, 185, 14, 13, 3, 3)$	44.6	46.3	68.8	421	14.3	252
$(\mathbf{128}, \mathbf{129}, \mathbf{175}, \mathbf{13}, \mathbf{14}, \mathbf{4}, \mathbf{0})$	39.3	106	111	364	13.7	270
$(128, 130, 175, 13, 14, 4, 3)$	39.1	60.3	106	364	13.7	270
$(128, 513, 265, 9, 9, 1, 0)$	156	562	35.1	1210	24.2	274
$(128, 514, 265, 9, 9, 1, 3)$	155	323	34.8	1210	24.1	274
$(128, 513, 202, 10, 11, 2, 0)$	58.5	658	46.4	547	16.4	234
$(128, 514, 202, 10, 11, 2, 3)$	59	389	46.3	547	16.4	234
$(128, 513, 183, 12, 12, 3, 0)$	44.1	567	66.5	408	14.5	243
$(128, 514, 183, 12, 12, 3, 3)$	44.7	326	68.4	408	14.5	243
$(\mathbf{128}, \mathbf{513}, \mathbf{174}, \mathbf{12}, \mathbf{12}, \mathbf{4}, \mathbf{0})$	38.5	750	82	352	13.4	258
$(128, 514, 174, 12, 12, 4, 3)$	38.3	418	80.4	352	13.4	258

$(\lambda, D, n, \Delta, v, \text{nb\_ite}, s)$	key gen. (MC)	sign (MC)	verify (KC)	$\|pk\|$ (KB)	$\|sk\|$ (KB)	sign (bits)
$(192, 17, 404, 20, 19, 1, 0)$	794	4.57	123	4300	57.8	423
$(192, 17, 310, 22, 23, 2, 0)$	267	5.12	154	2000	41.5	378
$(192, 17, 279, 23, 25, 3, 0)$	195	6.03	187	1480	37.4	400
$(\mathbf{192}, \mathbf{17}, \mathbf{266}, \mathbf{23}, \mathbf{25}, \mathbf{4}, \mathbf{0})$	171	8.38	255	1290	34.8	435
$(192, 33, 404, 20, 19, 1, 0)$	800	15.1	122	4300	59	423
$(192, 33, 310, 22, 23, 2, 0)$	271	16.3	155	2000	42.6	378
$(192, 33, 279, 23, 25, 3, 0)$	196	19.6	189	1480	38.4	400
$(192, 33, 266, 23, 25, 4, 0)$	174	27.2	255	1290	35.8	435
$(192, 129, 402, 18, 18, 1, 0)$	808	179	119	4240	59.6	420
$(192, 130, 402, 18, 18, 1, 3)$	813	115	119	4240	59.5	420
$(192, 640, 402, 18, 18, 1, 0)$	826	1620	120	4240	62.6	420
$(192, 640, 402, 18, 18, 1, 3)$	830	1100	120	4240	62.5	420
$(192, 129, 308, 20, 22, 2, 0)$	270	179	150	1970	43.1	372
$(192, 130, 308, 20, 22, 2, 3)$	269	117	151	1970	43.1	372
$(192, 129, 278, 22, 23, 3, 0)$	198	261	180	1450	38	391
$(192, 130, 278, 22, 23, 3, 3)$	196	157	182	1450	37.9	391
$(\mathbf{192}, \mathbf{129}, \mathbf{265}, \mathbf{22}, \mathbf{23}, \mathbf{4}, \mathbf{0})$	172	331	252	1260	35.4	423
$(192, 130, 265, 22, 23, 4, 3)$	173	202	249	1260	35.3	423
$(192, 513, 399, 15, 18, 1, 0)$	806	1280	117	4180	61.5	417
$(192, 514, 399, 15, 18, 1, 3)$	807	762	118	4180	61.4	417
$(192, 513, 308, 20, 19, 2, 0)$	272	1360	147	1930	41.7	366
$(192, 514, 308, 20, 19, 2, 3)$	273	721	146	1930	41.6	366
$(192, 513, 276, 20, 22, 3, 0)$	198	1840	181	1430	38.6	382
$(192, 514, 276, 20, 22, 3, 3)$	199	1070	180	1430	38.5	382
$(\mathbf{192}, \mathbf{513}, \mathbf{265}, \mathbf{22}, \mathbf{20}, \mathbf{4}, \mathbf{0})$	175	2320	239	1240	34.1	411
$(192, 514, 265, 22, 20, 4, 3)$	174	1260	233	1240	34	411

$(\lambda, D, n, \Delta, v, \text{nb\_ite}, s)$	key gen. (MC)	sign (MC)	verify (KC)	$\lvert pk \rvert$ (KB)	$\lvert sk \rvert$ (KB)	sign (bits)
$(256, 17, 540, 28, 29, 1, 0)$	2720	8.33	385	10400	117	569
$(256, 17, 415, 31, 32, 2, 0)$	959	9.77	363	4810	82.8	510
$(256, 17, 375, 33, 33, 3, 0)$	588	9.16	483	3570	73	540
$(\mathbf{256}, \mathbf{17}, \mathbf{358}, \mathbf{34}, \mathbf{35}, \mathbf{4}, \mathbf{0})$	523	12.9	588	3140	71.9	600
$(256, 33, 540, 28, 29, 1, 0)$	2740	27	383	10400	119	569
$(256, 33, 415, 31, 32, 2, 0)$	974	30.1	375	4810	84.7	510
$(256, 33, 375, 33, 33, 3, 0)$	602	29.2	488	3570	74.8	540
$(256, 33, 358, 34, 35, 4, 0)$	528	42.1	590	3140	73.7	600
$(256, 129, 540, 28, 26, 1, 0)$	2770	317	384	10300	116	566
$(256, 130, 540, 28, 26, 1, 3)$	2760	228	375	10300	116	566
$(256, 129, 414, 30, 30, 2, 0)$	971	379	359	4740	84.1	504
$(256, 130, 414, 30, 30, 2, 3)$	972	242	361	4740	84	504
$(256, 129, 372, 30, 33, 3, 0)$	600	407	471	3510	77.6	531
$(256, 130, 372, 30, 33, 3, 3)$	603	252	474	3510	77.5	531
$(\mathbf{256}, \mathbf{129}, \mathbf{358}, \mathbf{34}, \mathbf{32}, \mathbf{4}, \mathbf{0})$	529	545	583	3090	71.5	588
$(256, 130, 358, 34, 32, 4, 3)$	527	325	566	3090	71.4	588
$(256, 513, 537, 25, 26, 1, 0)$	2780	2700	379	10200	120	563
$(256, 514, 537, 25, 26, 1, 3)$	2770	1460	374	10200	120	563
$(256, 1152, 537, 25, 26, 1, 0)$	2810	7360	374	10200	123	563
$(256, 1152, 537, 25, 26, 1, 3)$	2800	4260	368	10200	123	563
$(256, 513, 414, 30, 27, 2, 0)$	970	2770	344	4680	81.7	498
$(256, 514, 414, 30, 27, 2, 3)$	983	1540	344	4680	81.6	498
$(256, 513, 372, 30, 30, 3, 0)$	603	3130	464	3460	75.3	522
$(256, 514, 372, 30, 30, 3, 3)$	601	1610	477	3460	75.2	522
$(\mathbf{256}, \mathbf{513}, \mathbf{354}, \mathbf{30}, \mathbf{33}, \mathbf{4}, \mathbf{0})$	532	3640	566	3040	75.9	576
$(256, 514, 354, 30, 33, 4, 3)$	524	2040	580	3040	75.8	576

Table 32: Performance of an exhaustive set of security parameters. We use a Skylake processor (LaptopS). The results have three significant digits. The parameters in bold correspond to RedGeMSS, BlueGeMSS and GeMSS.