Name of Proposal:

Gui

Principal Submitter:

Jintai Ding

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

Auxiliary Submitters: Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang

Inventors: c.f. Submitters

Owners: c.f. Submitters

Jintai Ding (Signature)

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

Nankang, Taipei 11529, Taiwan

Gui - Algorithm Specification and Documentation

Type: Signature scheme

Family: Multivariate Cryptography, BigField schemes

The Gui signature scheme as described in this proposal is based on the HFEv- signature scheme, which was first proposed by Patarin, Courtois and Goubin in [12]. Similar to Gui, their QUARTZ scheme uses a specially designed signature generation process which allows to reduce key and signature sizes compared to the original HFEv- design. However, while QUARTZ uses an HFE polynomial of high degree as well as small values for the numbers of minus equations and vinegar variables, Gui follows another approach. By decreasing the degree of the HFE polynomial in use while increasing the numbers of minus equations and vinegar variables, we can speed up the signature generation process of the scheme drastically without weakening its security.

1 Algorithm Specification

In this section we present the Gui signature scheme as proposed in [15].

1.1 Parameters

- $\mathbb{F} = \mathbb{F}_q$: finite field with q elements, $q = 2^e$
- $\mathbb{E} = \mathbb{F}_{q^n}$: degree n extension field of $\mathbb{F}$

- $\phi : \mathbb{F}^n \to \mathbb{E}$: isomorphism between the vector space $\mathbb{F}^n$ and the extension field $\mathbb{E}$
- D: degree of the HFE polynomial, set $r = \lfloor \log_q(D-1) \rfloor + 1$.
- a: number of minus equations
- v: number of vinegar variables
- k: repetition factor (used in signature generation)
- number of equations: $n - a$
- number of variables: $n + v$

1.2 Key Generation

Private Key. The private key consists of the three maps

- $\mathcal{S} : \mathbb{F}^n \to \mathbb{F}^{n-a}$: affine transformation of maximal rank
- $\mathcal{T} : \mathbb{F}^{n+v} \to \mathbb{F}^{n+v}$: invertible affine transformation
- central map $\mathcal{F} : \mathbb{E} \times \mathbb{F}^v \to \mathbb{E}$

$$\mathcal{F}(X) = \sum_{0 \le i \le j}^{q^i+q^j \le D} \alpha_{i,j} X^{q^i+q^j} + \sum_{0 \le i}^{q^i \le D} \beta_i(v_1, \ldots, v_v) \cdot X^{q^i} + \gamma(v_1, \ldots, v_v). \quad (1)$$

Here, the $\beta_i : \mathbb{F}^v \to \mathbb{E}$ are linear (affine) functions in the vinegar variables $v_1, \ldots, v_v$, while $\gamma : \mathbb{F}^v \to \mathbb{E}$ is a quadratic function in $v_1, \ldots, v_v$.

Due to the special structure of $\mathcal{F}$, the map $\bar{\mathcal{F}} = \phi^{-1} \circ \mathcal{F} \circ (\phi \times id_v)$ is a quadratic multivariate map from $\mathbb{F}^{n+v}$ to $\mathbb{F}^n$.

The size of the private key is

$$\underbrace{(n-a)\cdot(n+1)}_{\text{affine map } \mathcal{S}} + \underbrace{(n+v)\cdot(n+v+1)}_{\text{affine map } \mathcal{T}} + \underbrace{n \cdot \left(\#\alpha + (\lfloor \log_q D \rfloor + 1)\cdot(v+1) + \frac{(v+1)\cdot(v+2)}{2} \right)}_{\text{central map } \mathcal{F}}$$

$\mathbb{F}_q$-elements. Here, $\#\alpha$ denotes the number of non-zero coefficients α in equation (1), which is upper bounded by $\frac{r\cdot(r+1)}{2}$.

Public Key. The public key of Gui is the composed map

$$\mathcal{P} = \mathcal{S} \circ \phi^{-1} \circ \mathcal{F} \circ (\phi \times id_v) \circ \mathcal{T} : \mathbb{F}^{n+v} \to \mathbb{F}^{n-a}.$$

consisting of $n-a$ quadratic poltynomials in $n+v$ variables. The size of the public key is

$$(n-a) \cdot \frac{(n+v+1) \cdot (n+v+2)}{2}$$

$\mathbb{F}_q$-elements.

1.3 Signature Generation

Given a document d to be signed, we first compute hash values $\mathbf{d}_1, \ldots, \mathbf{d}_k \in \mathbb{F}_q^{n-a}$ as follows. For a standard hash function $\mathcal{H} : \{0,1\}^\star \to \{0,1\}^{n'}$ we compute a bitstring

$$\tilde{\mathbf{h}} = \mathcal{H}(d)||\mathcal{H}(\mathcal{H}(d))|| \ldots ||\mathcal{H}^\ell(d)$$

of length $k \cdot \log_2 q \cdot (n-a) \leq |\tilde{\mathbf{h}}| < (k+1) \cdot \log_2 q \cdot (n-a)$. Here, $\mathcal{H}^\ell(d)$ denotes the ℓ-times repeated application of the hash function $\mathcal{H}$. We set

$$\mathbf{d}_i = (\tilde{\mathbf{h}}_{(i-1)\cdot\log_2 q\cdot(n-a)+1}|| \ldots ||\tilde{\mathbf{h}}_{i\cdot\log_2 q\cdot(n-a)}) \quad (i = 1, \ldots, k)$$

and transform each $\mathbf{d}_i$ into a vector of $(n-a)$ $\mathbb{F}_q$-elements.
The last $\ell \cdot n' - k \cdot \log_2 q \cdot (n-a)$ bits of $\tilde{\mathbf{h}}$ are skipped.

After having computed the hash values $\mathbf{d}_i \in \mathbb{F}^{n-a}$ $(i = 1, \ldots, k)$, we obtain a Gui signature for the message d as follows.

We set $S_0 = \mathbf{0}^{n-a}$ and perform for $i = 1$ to k the following steps

1. Compute a preimage $\mathbf{x} \in \mathbb{F}^n$ of $\mathbf{d}_i \oplus S_{i-1}$ under the affine map $\mathcal{S}$ and lift it to the extension field, obtaining $X \in \mathbb{E}$.

2. Choose random values for the vinegar variables $v_1, \ldots, v_v$ and substitute them into the central map to obtain the parametrized map $\mathcal{F}_V : \mathbb{E} \to \mathbb{E}$.

3. Find a solution to the equation $\mathcal{F}_V(Y) = X$ by performing the first step of the Cantor-Zassenhaus algorithm, i.e. compute $\tilde{Y} = \gcd(\mathcal{F}_V(\hat{Y}) - X, \hat{Y}^{2^n} - \hat{Y})$. For efficiency reasons, we repeat step 2 and 3 until the gcd is a linear polynomial. Denote the unique root of $\tilde{Y}$ by $Y \in \mathbb{E}$.

4. Move Y down to the vector space $\mathbb{F}^n$, obtaining $\mathbf{y}' = (y_1, \ldots, y_n)$, and append the vinegar variables of step 2, obtaining $\mathbf{y} = (\mathbf{y}'||v_1|| \ldots ||v_v) \in \mathbb{F}^{n+v}$.

5. Compute $\mathbf{z} = (z_1, \ldots, z_{n+v}) = \mathcal{T}^{-1}(\mathbf{y})$ and set $S_i = (z_1, \ldots, z_{n-a})$, $X_i = (z_{n-a+1}, \ldots, z_{n+v})$.

The final Gui signature of the message d has the form $\sigma = (S_k||X_k|| \ldots ||X_1) \in \mathbb{F}^{(n-a)+k\cdot(a+v)}$.

1.4 Signature Verification

In order to check the authenticity of a signature $\sigma = (S_k||X_k||\ldots||X_1) \in \mathbb{F}^{(n-a)+k\cdot(a+v)}$, we first compute the hash values $\mathbf{d}_i \;\; (i = 1, \ldots, k)$ as described in the previous section.
After that, we perform for $i = k-1, \ldots, 0$ the two steps

1. Evaluate the public key at $(S_{i+1}||X_{i+1})$. Denote the result by $\mathbf{w} \in \mathbb{F}^{n-a}$.
2. Set $S_i = \mathbf{w} \oplus \mathbf{d}_{i+1}$.

The signature σ is accepted, if and only if $S_0 = \mathbf{0}^{n-a}$ holds.

In order to find a good balance between security and efficiency, the Gui (and HFEv-) signature scheme is mainly used over small finite fields. Since the complexity of the signature generation process is $O(n \cdot D^3)$, one aims at choosing the degree D of the HFE polynomial in use as small as possible. On the other hand, for security reasons, the number of quadratic terms in the HFE polynomial (1) and therefore the value $r = \lfloor \log_q(D-1) \rfloor + 1$ should not be too small. For a fixed D, we therefore can increase r and the number of quadratic terms in the HFEv-polynomial by simply decreasing q. So, a small value of q allows to achieve both good performance and high security. In order to reduce the number of equations in the public system and therefore key and signature sizes, one introduces a repetition factor k. To reach a security level of ℓ bits under collision attacks, one would need, for the pure HFEv- scheme ($k = 1$), $2 \cdot \ell / \log_2(q)$ equations in the public key, leading to a very large public key. By choosing $k > 1$, we can omit this problem by using a hash of effective length $k \cdot |\mathcal{H}|$. However, choosing $k > 2$ brings no advantage, but increases the signature size.

The following algorithms `GuiKeyGen`, `GuiSign` and `GuiVer` show the key generation, signature generation and signature verification processes of Gui in algorithmic form.

Algorithm 1 `GuiKeyGen`: Key Generation of Gui

Input: Gui parameters (q, n, D, a, v), isomorphism $\phi : \mathbb{F}_q^n \to \mathbb{E}$
Output: Gui key pair (sk, pk)

repeat
  $M_S \leftarrow \texttt{Matrix}(q, n, n)$
until IsInvertible(M_S) == **TRUE**
$c_S \leftarrow_R \mathbb{F}^n$
$\mathcal{S} \leftarrow \texttt{Aff}(M_S, c_S)$
$InvS \leftarrow M_S^{-1}$
repeat
  $M_T \leftarrow \texttt{Matrix}(q, n+v, n+v)$
until IsInvertible(M_T) == **TRUE**
$c_T \leftarrow_R \mathbb{F}^{n+v}$
$\mathcal{T} \leftarrow \texttt{Aff}(M_T, c_T)$
$InvT \leftarrow M_T^{-1}$
$\mathcal{F} \leftarrow \texttt{HFEvmap}(q, n, D, a, v)$
$\mathcal{P} \leftarrow \mathcal{S} \circ \phi^{-1} \circ \mathcal{F} \circ (\phi \times id_v) \circ \mathcal{T}$
$sk \leftarrow (InvS, c_S, \mathcal{F}, InvT, c_T)$
$pk \leftarrow \mathcal{P}$
return (sk, pk)

The possible input values of algorithm `GuiKeyGen` are specified in Section 1.8. The function $\texttt{Matrix}(q, m, n)$ returns an $m \times n$ matrix with coefficients chosen uniformly at random in $\mathbb{F}_q$. $\texttt{Aff}(M, c)$ returns the affine map $M \cdot x + c$. $\texttt{HFEvmap}(q, D, a, v, \beta_i, \gamma)$ outputs an HFEv central map (see equation (1)) with randomly chosen coefficients $\alpha \in \mathbb{E}$ and randomly chosen vinegar maps β_i $(i = 0, \ldots, \lfloor \log_q D \rfloor)$ and γ.

The algorithm `GuiKeyGen` makes use of

$$\underbrace{n \cdot (n+1)}_{\text{affine map } \mathcal{S}} + \underbrace{(n+v) \cdot (n+v+1)}_{\text{affine map } \mathcal{T}} + \underbrace{n \cdot \left(\#\alpha + (\lfloor \log_q D \rfloor + 1) \cdot (v+1) + \frac{(v+1) \cdot (v+2)}{2} \right)}_{\text{central map } \mathcal{F}}$$

randomly generated field elements. Here, $\#\alpha$ denotes the number of non-zero coefficients α in equation (1), which is upper bounded by $\frac{r \cdot (r+1)}{2}$. In contrast to the description in Section 1.2, we use here and in our implementation an invertible matrix $M_S \in \mathbb{F}_q^{n \times n}$. While this increases the private key size slightly, it speeds up the signature generation significantly.

Algorithm 2 `GuiSign`: Signature Generation Process of Gui

Input: Gui private key ($InvS$, c_S, $\mathcal{F}$, $InvT$, c_T), message d, repetition factor k

Output: signature $\sigma \in \mathbb{F}^{(n-a)+k\cdot(a+v)}$

1: $\ell \leftarrow \lceil k \cdot \log_2 q \cdot (n-a)/|\mathcal{H}| \rceil$
2: $\tilde{\mathbf{h}} \leftarrow \mathcal{H}(d)||\mathcal{H}(\mathcal{H}(d))|| \ldots ||\mathcal{H}^{\ell}(d)$
3: $S_0 \leftarrow \mathbf{0}^{n-a}$
4: **for** $i = 1$ to k **do**
5: $\quad \mathbf{d}_i \leftarrow \mathbb{F}^{n-a}!(\tilde{\mathbf{h}}_{(i-1)\cdot\log_2 q\cdot(n-a)+1}, \ldots, \tilde{\mathbf{h}}_{i\cdot\log_2 q\cdot(n-a)})$
6: $\quad (S_i, X_i) \leftarrow$ `InvHFEv-`$(\mathbf{d}_i \oplus S_{i-1})$
7: **end for**
8: $\sigma \leftarrow (S_k||X_k|| \ldots ||X_1)$
9: **return** σ

Algorithm 3 `InvHFEv-`: Inversion of the HFEv- public key

Input: Gui private key ($InvS$, c_S, $\mathcal{F}$, $InvT$, c_T), isomorphism $\phi : \mathbb{F}_q^n \to \mathbb{E}$, vector $\mathbf{w} \in \mathbb{F}^{n-a}$,

Output: vector $\mathbf{z} \in \mathbb{F}^{n+v}$ such that $\mathcal{P}(\mathbf{z}) = \mathbf{w}$

1: $r_1, \ldots, r_a \leftarrow_R \mathbb{F}$
2: $\mathbf{x} \leftarrow InvS \cdot ((\mathbf{w}||r_1|| \ldots ||r_a) - c_S)$
3: $X \leftarrow \phi(\mathbf{x})$
4: **repeat**
5: $\quad v_1, \ldots, v_v \leftarrow_R \mathbb{F}$
6: $\quad \mathcal{F}_V \leftarrow \mathcal{F}(v_1, \ldots, v_v)$
7: $\quad Y \leftarrow \gcd(\mathcal{F}_V(\hat{Y}) - X, \hat{Y}^{2^n} - \hat{Y})$
8: **until** $\deg(Y) == 1$
9: $\mathbf{y} \leftarrow \phi^{-1}(\texttt{root}(Y))$
10: $\mathbf{z} \leftarrow InvT \cdot ((\mathbf{y}||v_1|| \ldots ||v_v) - c_T)$
11: **return z**

In line 1 of Algorithm `GuiSign`, $|\mathcal{H}|$ denotes the output length of the hash function in use. $\mathbb{F}^{n-a}!(\mathbf{h})$ coerces the binary vector $\mathbf{h} \in \mathrm{GF}(2)^{\log_2 q\cdot(n-a)}$ into a vector in $\mathbb{F}^{n-a}$. [1]

In Algorithm `InvHFEv-`, we perform the loop (line 4 to 8) until the computed gcd is a linear polynomial (i.e. if the equation $\mathcal{F}_V(\hat{Y}) = X$ has a unique solution). In this case, the function `root`(Y) returns the unique root of the linear polynomial $Y \in \mathbb{E}[\hat{Y}]$. Otherwise, we choose other values for the vinegar variables $v_1, \ldots, v_v$ and try again. Though this check requires more gcd computations, it is still more efficient than computing the roots of a polynomial Y of higher degree.

During the signature generation of Gui, we make use of approximately $k\cdot(a+e\cdot v)$ randomly chosen field elements (in line 2 and 6 of algorithm `InvHFEv-`). The reason for this is that, in order to find a unique solution of $\mathcal{F}_V(\hat{Y}) = X$ (see

[1] In the case of $\mathbb{F} = \mathrm{GF}(2)$, this function does not do anything.

line 8 of Algorithm `InvHFEv-`), we have to run the loop about e times, while the whole algorithm is performed k times..

Algorithm 4 `GuiVer`: Signature Verification Process of Gui

Input: Gui public key $\mathcal{P}$, message d, repetition factor k, signature $\sigma \in \mathbb{F}^{(n-a)+k(a+v)}$

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

1: $\ell \leftarrow \lceil k \cdot \log_2 q \cdot (n-a)/|\mathcal{H}| \rceil$
2: $\tilde{\mathbf{h}} \leftarrow \mathcal{H}(d)||\mathcal{H}(\mathcal{H}(d))|| \ldots ||\mathcal{H}^{\ell}(d)$
3: **for** $i = 1$ to k do **do**
4: $\quad \mathbf{d}_i \leftarrow \mathbb{F}^{n-a}!(\tilde{\mathbf{h}}_{(i-1)\cdot\log_2 q\cdot(n-a)+1}, \ldots, \tilde{\mathbf{h}}_{i\cdot\log_2 q\cdot(n-a)})$
5: **end for**
6: **for** $i = k-1$ to 0 **do**
7: $\quad S_i \leftarrow \mathcal{P}(S_{i+1}||X_{i+1}) \oplus \mathbf{d}_{i+1}$
8: **end for**
9: **if** $S_0 = \mathbf{0}$ **then**
10: $\quad$ **return TRUE**
11: **else**
12: $\quad$ **return FALSE**
13: **end if**

1.5 Remark on Correctness

In the signature generation process, we start with $S_0 = \mathbf{0}$ and set recursively $(S_i||X_i) = \texttt{InvHFEv}-(\mathbf{d}_i \oplus S_{i-1})$ until we finally obtain the signature $\sigma = (S_k||X_k|| \ldots ||X_1)$.
During the signature verification process, we start with $\tilde{S}_k = S_k$ and compute recursively $\tilde{S}_{i-1} \leftarrow \texttt{HFEv}-(\tilde{S}_i||X_i) \oplus \mathbf{d}_i$ until we get $\tilde{S}_0$. We find

$$\begin{aligned} \tilde{S}_{k-1} &= \texttt{HFEv}-(S_k||X_k) \oplus \mathbf{d}_k = \texttt{HFEv}-(\texttt{InvHFEv}-(\mathbf{d}_k \oplus S_{k-1})) \oplus \mathbf{d}_k \\ &= \mathbf{d}_k \oplus S_{k-1} \oplus \mathbf{d}_k = S_{k-1}. \end{aligned}$$

Analogously we obtain $\tilde{S}_{k-2} = S_{k-2}, \ldots, \tilde{S}_0 = S_0 = \mathbf{0} \in \mathbb{F}^{n-a}$. Therefore, a honestly generated signature will always be accepted.

1.6 Changes needed to achieve EUF-CMA Security

The standard Gui signature scheme as described above provides only universal unforgeability. In order to obtain EUF-CMA security, we apply a transformation similar to that in [16]. The main difference is the use of a random binary vector r called salt. Instead of generating a signature for the hash value $\mathbf{h} = \mathcal{H}(d)$, we generate a signature for $\mathcal{H}(\mathcal{H}(d)||r)$. The resulting signature has the form $\sigma^\star = (\sigma, r)$, where σ is a standard Gui signature. By doing so, we ensure that an attacker is not able to forge any hash/signature pair.

In particular, we apply the following changes to the algorithms `GuiKeyGen`, `GuiSign` and `GuiVer`.

- In the algorithm `GuiKeyGen`*, we choose an integer $\bar{\ell}$ as the length of the random salt; $\bar{\ell}$ is appended both to the private and public key.
- In the algorithm `GuiSign`*, we choose first randomly the values of the $k \cdot (a + v)$ random variables (the random values for the affine map $\mathcal{S}$ and the vinegar variables of the central map); after that, we choose a random salt $r \in \{0,1\}^{\bar{\ell}}$ and perfom the standard Gui signature generation process to obtain a signature $\sigma^\star = (S_k||X_k|| \ldots ||X_1||r)$ for the message $\mathcal{H}(d)||r$. If there appears an error (i.e. one of the equations $\mathcal{F}_V(\hat{Y}) = X$ is not uniquely solvable), we choose a new value for the salt r and try again.
- The verification algorithm `GuiVer`* returns **TRUE** if we obtain $S_0 = \mathbf{0}^{n-a}$, and **FALSE** otherwise

Algorithms `GuiKeyGen`*, `GuiSign`* and `GuiVer`* show the modified key generation, signing and verification algorithms.

Algorithm 5 `KeyGen`*: Modified Key Generation Algorithm for Gui

Input: Gui parameters (q, n, D, a, v), length $\bar{\ell}$ of the random salt
Output: Gui key pair (sk, pk)

1: $sk, pk \leftarrow$ `GuiKeyGen`(q, n, D, a, v)
2: $sk \leftarrow sk,\ \bar{\ell}$
3: $pk \leftarrow pk,\ \bar{\ell}$
4: **return** (sk, pk)

The value of $\bar{\ell}$ is specified at the end of this section.

Algorithm 6 `GuiSign`*: Modified signature generation process for Gui

Input: document d, Gui private key $(InvS, c_S, \mathcal{F}, InvT, c_T)$, length $\bar{\ell}$ of salt
Output: Gui signature $\sigma = (S_k||X_k||\dots||X_1||r) \in \mathbb{F}^{(n-a)+k\cdot(a+v)} \times \{0,1\}^{\bar{\ell}}$

1: $\ell \leftarrow \lceil k \cdot \log_2 q \cdot (n-a)/|\mathcal{H}| \rceil$
2: $r_1^{(1)}, \dots, r_a^{(1)}, r_1^{(2)}, \dots, r_a^{(k)}, v_1^{(1)}, \dots, v_v^{(1)}, v_1^{(2)}, \dots, v_v^{(k)} \leftarrow_R \mathbb{F}$
3: **repeat**
4: $\quad S_0 \leftarrow \mathbf{0}$
5: $\quad r \leftarrow \{0,1\}^{\bar{\ell}}$
6: $\quad \tilde{\mathbf{h}} \leftarrow \mathcal{H}(\mathcal{H}(d)||r)||\mathcal{H}(\mathcal{H}(\mathcal{H}(d)||r))||\dots||\mathcal{H}^\ell(\mathcal{H}(d)||r)$
7: $\quad$ **for** $i = 1$ to k do **do**
8: $\quad\quad \mathbf{d}_i \leftarrow \mathbb{F}^{n-a}!(\tilde{\mathbf{h}}_{(i-1)\cdot\log_2 q\cdot(n-a)+1}, \dots, \tilde{\mathbf{h}}_{i\cdot\log_2 q\cdot(n-a)})$
9: $\quad\quad t, S_i, X_i \leftarrow$ `InvHFEv`$-^\star(\mathbf{d}_i \oplus S_{i-1}, r_1^{(i)}, \dots, r_a^{(i)}, v_1^{(i)}, \dots, v_v^{(i)})$
10: $\quad\quad$ **if** $t ==$ **FALSE then**
11: $\quad\quad\quad$ **break** and **go to** 4
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: **until** $t ==$ **TRUE**
15: $\sigma^\star \leftarrow (S_k||X_k||\dots||X_1||r)$
16: **return** $\sigma^\star$

Note that, in algorithm `GuiSign`* we do not generate a signature for $\mathcal{H}(d||r)$, but for $\mathcal{H}(\mathcal{H}(d)||r)$. In case we have to run the loop in the algorithm several times, this speeds up the signature generation of our scheme significantly (at least for long messages d).
In algorithm `GuiSign`*, we make use of approximately $k \cdot \log_2 q \cdot (a+v) + e \cdot k \cdot \bar{\ell}$ random bits.

Algorithm 7 `InvHFEv-`*: Modified Inversion of the HFEv- public key

Input: Gui private key $(InvS, c_S, \mathcal{F}, InvT, c_T)$, isomorphism $\phi : \mathbb{F}^n \to \mathbb{E}$,
hash value $\mathbf{w} \in \mathbb{F}^{n-a}$, random values $r_1, \dots, r_a$, vinegar values $v_1, \dots, v_v$
Output: boolean value t, vector $\mathbf{z} \in \mathbb{F}^{n+v}$

1: $\ell \to \lceil k \cdot \log_2 \cdot (n-a)/|\mathcal{H}| \rceil$
2: $\mathbf{x} \leftarrow InvS \cdot ((\mathbf{w}||r_1||\dots||r_a) - c_S)$
3: $X \leftarrow \phi(\mathbf{x})$
4: $\mathcal{F}_V \leftarrow \mathcal{F}(v_1, \dots, v_v)$
5: $Y \leftarrow \gcd(\mathcal{F}_V(\hat{Y}) - X, \hat{Y}^{2^n} - \hat{Y})$
6: **if** $\deg(Y) == 1$ **then**
7: $\quad \mathbf{y} \leftarrow \phi^{-1}(\mathtt{root}(Y))$
8: $\quad \mathbf{z} \leftarrow InvT((\mathbf{y}||v_1||\dots||v_v) - c_T)$
9: $\quad$ **return TRUE**, $\mathbf{z}$
10: **else**
11: $\quad$ **return FALSE**, $\mathbf{0}^{n+v}$
12: **end if**

In line 6 of algorithm `InvHFEv`$-^\star$ we check if the computed gcd is a linear

polynomial in $Y \in \mathbb{E}[\hat{Y}]$. If so, the function $\texttt{root}(Y)$ returns the unique root of Y and the algorithm returns **TRUE** as well as the unique solution of $\mathcal{P}(\mathbf{z}) = \mathbf{w}$. Otherwise, the algorithm returns **FALSE** (and the zero vector $\mathbf{0} \in \mathbb{F}^{n+v}$). Though this check makes it necessary to run algorithm $\texttt{InvHFEv}-^{\star}$ more often, it is still more efficient than computing the root of a polynomial Y of higher degree.

Algorithm 8 $\texttt{GuiVer}^{\star}$: Modified signature verification process for Gui

Input: Gui public key $\mathcal{P}$, document d, signature $\sigma = (S_k||X_k|| \dots ||X_1||r) \in \mathbb{F}^{(n-a)+k\cdot(a+v)} \times \{0,1\}^{\ell}$

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

$\ell \leftarrow \lceil k \cdot \log_2 q \cdot (n-a)/|\mathcal{H}| \rceil$
$\tilde{\mathbf{h}} \leftarrow \mathcal{H}(\mathcal{H}(d)||r)||\mathcal{H}(\mathcal{H}(\mathcal{H}(d)||r))|| \dots ||\mathcal{H}^{\ell}(\mathcal{H}(d)||r)$
for $i = 1$ to k do **do**
    $\mathbf{d}_i \leftarrow \mathbb{F}^{n-a}!(\tilde{\mathbf{h}}_{(i-1)\cdot\log_2 q\cdot(n-a)+1}, \dots, \tilde{\mathbf{h}}_{i\cdot\log_2 q\cdot(n-a)})$
end for
for $i = k-1$ to 0 **do**
    $S_i \leftarrow \mathcal{P}(S_{i+1}||X_{i+1}) \oplus \mathbf{d}_{i+1}$
end for
if $S_0 = \mathbf{0}$ **then**
    return TRUE
else
    return FALSE
end if

Similar to [16] we find that every attacker, who can break the EUF-CMA security of the modified scheme, can also break the standard Gui signature scheme.

In order to get a secure scheme, we must ensure that every signature is generated using a different random seed. Under the assumption of maximal 2^{64} signatures being generated with the system [11], a random salt of length 128 bit seems reasonable.

1.7 Note on the generation of random field elements

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

1.8 Parameter Choice

We choose GF(2) as the underlying finite field and choose the repetition factor k to be 2. We propose the following three parameter sets for Gui

Gui-184 Gui(GF(2),184,33,16,16,2) with 168 equations in 200 variables

Gui-312 Gui(GF(2),312,129,24,20,2) with 288 equations in 332 variables

Gui-448 Gui(GF(2),448,513,32,28,2) with 416 equations in 476 variables

The reason for restricting our scheme to the field GF(2) is to find a good balance between security and efficiency. During the signature generation process, we have to invert a univariate polynomial of degree D over the extension field $\mathbb{E}$. The complexity of this step can be estimated as

$$\text{Complexity}_{\texttt{SignGen}} = O(\log_q(|\mathbb{E}|) \cdot D^3) = O(n \cdot D^3).$$

For efficiency reasons, we therefore aim at decreasing D as far as possible.
On the other hand we want, for security reasons, the HFE polynomial (1) to contain not too few quadratic terms. Since this number directly depends on $r = \lfloor \log_q(D-1) \rfloor + 1$, we choose q as small as possible, i.e $q = 2$.
The idea behind the choice of D, a and v is that we want HFE, the minus method and the vinegar modification to play a more or less equally important role in enhancing the security of our scheme. Therefore, we choose D, a and v to increase more or less proportionally with increasing security. By doing so, Gui can be seen as balanced version of HFEv-, where HFE, the minus method and the vinegar modification are equally important. Due to the vast improvements in implementation efficiency, our solution comes without much sacrifice of efficiency, but reduces the risk of possible future attacks against Gui significantly.
For ease of implementation, we furthermore choose the parameters n, a and v of Gui in such a way that the lengths of the hash values $\mathbf{d}_i$ $(i = 1, \ldots, k)$ and the resulting Gui signatures are multiples of 8 bit. Moreover, for efficiency reasons (special processor instructions), the size n of the extension fields is chosen to be close to multiples of 64.
The resulting key and signature sizes can be found in Section 4.1.

Additionally to the three parameter proposals Gui-184, Gui-312 and Gui-448, we give here two more parameter sets to illustrate certain aspects of the parameter choice.

Gui-160 Gui(GF(2),160,33,16,16,2) The value $n = 160$ is not large enough to prevent quantum brute force attacks (see Section attacks 6.2) against the scheme. While the complexity of a classical brute force attack of 2^{147} gates is acceptable for NIST security category I, the complexity of a quantum brute force attackagainst the scheme is only 2^{94} (logical) quantum gates.

Gui-192 Gui(GF(2), 192,9,8,8,2) While the number of equations in the scheme is large enough to prevent brute force attacks, the scheme is, due to the small values of D, a and v, vulnerable by direct attacks (see Section 6.3). The degree of regularity of a direct attack against this scheme can be estimated as $(4 + 8 + 8 + 7)/3 = 9$, leading to a complexity of the direct attack of about 2^{116} gates.

1.8.1 Note on the used hash functions

We use SHA-2 as the hash function underlying our Gui instances. The SHA-2 hash function family comprises the four hash functions SHA224, SHA256, SHA384 and SHA512 with output lengths of 224, 256, 384 and 512 bits respectively. We use

- SHA256 for Gui-184
- SHA-384 for Gui-312 and
- SHA-512 for Gui-448.

An analysis of the security of the proposed Gui instances against collision attacks can be found in Section 5.3.

2 Key Storage

2.1 Representation of Finite Field Elements

2.1.1 Elements of $\mathbb{F}_2$

The field of two elements, denoted as $\mathbb{F}_2$, is the set $\{0, 1\}$. The multiplication of elements in $\mathbb{F}_2$ is a logic `AND` and addition is a logic `XOR`. Each element of $\mathbb{F}_2$ is stored in one bit.

2.1.2 Vectors over $\mathbb{F}_2$

A vector of l elements in $\mathbb{F}_2$ is represented as a bit sequence of length l. Vectors are the basic building blocks of our implementations. All components in the key file are in the form of vectors over $\mathbb{F}_2$ of different length.
Suppose $\mathbf{v} = (v_0, \ldots, v_{l-1}) \in \mathbb{F}_2^l$ and $v_i \in \mathbb{F}_2$.

- v_0 corresponds to the least significant bit of the l bit sequence.
- If l is a multiple of 8, $\mathbf{v}$ is stored in $l/8$ bytes.
- If l is not a multiple of 8, $\mathbf{v}$ is stored in $\lfloor (l + 7)/8 \rfloor$ bytes and the bits with index $> l$ are padded with 0.

2.2 Public Key

The public key $\mathcal{P}$ of Gui is a set of m multivariate quadratic polynomials in n variables (we write $\mathcal{P} := \mathcal{MQ}(m, n)$). The monomials are ordered according to the "graded-reverse-lexicographic" order.
Therefore, the system $\mathcal{P}$ looks as follows.

$$
\begin{aligned}
y_1 &= q_{2,1,1} \cdot x_2 x_1 + q_{3,1,1} \cdot x_3 x_1 + q_{3,2,1} \cdot x_3 x_2 + q_{4,1,1} \cdot x_4 x_1 + q_{4,2,1} \cdot x_4 x_2 \\
&\quad + q_{4,3,1} \cdot x_4 x_3 + \cdots + q_{n,n-1,1} \cdot x_n x_{n-1} + l_{1,1} x_1 + l_{2,1} x_2 + \cdots + l_{n,1} x_n + c_1 \\
y_2 &= q_{2,1,2} \cdot x_2 x_1 + q_{3,1,2} \cdot x_3 x_1 + q_{3,2,2} \cdot x_3 x_2 + q_{4,1,2} \cdot x_4 x_1 + q_{4,2,2} \cdot x_4 x_2 \\
&\quad + q_{4,3,2} \cdot x_4 x_3 + \cdots + q_{n,n-1,2} \cdot x_n x_{n-1} + l_{1,2} x_1 + l_{2,2} x_2 + \cdots + l_{n,2} x_n + c_2 \\
&\quad \vdots \\
y_m &= q_{2,1,m} \cdot x_2 x_1 + q_{3,1,m} \cdot x_3 x_1 + q_{3,2,m} \cdot x_3 x_2 + q_{4,1,m} \cdot x_4 x_1 + q_{4,2,m} \cdot x_4 x_2 \\
&\quad + q_{4,3,m} \cdot x_4 x_3 + \cdots + q_{n,n-1,m} \cdot x_n x_{n-1} + l_{1,m} x_1 + l_{2,m} x_2 + \cdots + l_{n,m} x_n + c_m
\end{aligned}
$$

Here, $q_{i,j,k}$ is the coefficient of the quadratic monomial $x_i x_j$ of the polynomial y_k, $l_{i,k}$ the coefficient of the linear monomial x_i in y_k and c_k the constant coefficient of y_k ($1 \leq j < i \leq n, 1 \leq k \leq m$). Note that there are no $x_i x_j$ terms with $i = j$, since the polynomials are defined over $\mathbb{F}_2$.
We define the vectors $\mathbf{q}_{i,j} = (q_{i,j,k})_{k=1}^{m} \in \mathbb{F}_2^m$, $\mathbf{l}_i = (l_{i,k})_{k=1}^{m} \in \mathbb{F}_2^m$, and $\mathbf{c} = (c_1, \ldots, c_m) \in \mathbb{F}_2^m$. Using this notation, the public key $\mathcal{P}$ is stored as a byte sequence

$$[\mathbf{l}_1, \ldots, \mathbf{l}_n, \mathbf{q}_{2,1}, \mathbf{q}_{3,1}, \mathbf{q}_{3,2}, \ldots, \mathbf{q}_{n,n-2}, \mathbf{q}_{n,n-1}, \mathbf{c}].$$

2.3 Secret Key

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

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

Suppose the affine map $\mathcal{T}(\mathbf{x}) : \mathbb{F}^{n+v} \to \mathbb{F}^{n+v}$ is given by

$$\mathcal{T}(\mathbf{x}) = \begin{bmatrix} t_{1,1} & t_{1,2} & \ldots & t_{1,n+v} \\ & \vdots & \ddots & \\ t_{n+v,1} & t_{n+v,2} & \ldots & t_{n+v,n+v} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{n+v} \end{bmatrix} + \begin{bmatrix} c_1 \\ \vdots \\ c_{n+v} \end{bmatrix}.$$

We store the matrix in column-major form. Define, for $i := 1, \ldots, n+v$, the column vector $\mathbf{t}_i = (t_{1,i}, \ldots, t_{n+v,i}) \in \mathbb{F}_2^{n+v}$, as well as the vector of the constant terms $\mathbf{c} = (c_1, \ldots, c_{n+v}) \in \mathbb{F}_2^{n+v}$. With this, the affine map $\mathcal{T}$ is stored as the sequence

$$[\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{n'}, \mathbf{c}].$$

The affine map $\mathcal{S}$ is stored in the same manner.

2.3.2 The central map $\mathcal{F}$

Recall that the central map of Gui is a map from $\mathbb{E} \times \mathbb{F}^v$ to $\mathbb{E}$ of the form

$$\mathcal{F}(X) = \sum_{\substack{0 \leq i \leq j \\ q^i + q^j \leq D}} \alpha_{i,j} X^{q^i + q^j} + \sum_{\substack{0 \leq i \\ q^i \leq D}} \beta_i(v_1, \ldots, v_v) X^{q^i} + \gamma(v_1, \ldots, v_v).$$

An element in $\mathbb{E} := \mathbb{F}_{2^n}$ is stored as an n dimensional vector in $\mathbb{F}_2^n$. The field isomorphism $\phi : \mathbb{F}_2^n \to \mathbb{E}$ is described in the next section.
The coefficients of $\mathcal{F}(X)$ are stored in the order $\alpha_{i,j}$, β_i, and γ.

$\alpha_{i,j}$: The coefficients $(\alpha_{i,j})_{0\le i\le j}^{2^i+2^j\le D} \in \mathbb{E}$ are divided into 2 groups $A : (\alpha_{i,j})_{0\le i=j}^{2^i+2^j\le D}$ and $B : (\alpha_{i,j})_{0\le i<j}^{2^i+2^j\le D}$. These elements are stored in the order A, B. Within A and B, the coefficients $\alpha_{i,j}$ are stored in ascending order (with respect to the exponent $2^i + 2^j$).
The reason for partitioning the coefficients $\alpha_{i,j}$ into the two groupt A and B is that the elements of A a later combined with the maps β_i to compute the coefficients of the linear terms of the parametrized map $\mathcal{F}_V$.

$\beta_i(v_1, \ldots, v_v)$: Each β_i contains v $\mathbb{E}$-elements corresponding to the coefficients of the linear monomials $v_1, \ldots, v_v$. We denote the elements of the vector β_i by $(\beta_{i,1}, \ldots, \beta_{i,v})$. The β part of $\mathcal{F}(X)$ is stored in the order

$$[\beta_{0,1}, \ldots, \beta_{0,v}, \beta_{1,1}, \ldots, \beta_{\lfloor \log_2 D \rfloor, v}].$$

$\gamma(v_1, \ldots, v_v)$: The map $\gamma : \mathbb{F}^v \to \mathbb{E}$ is a multivariate quadratic polynomial in the n variables $v_1, \ldots, v_n$. Using the field equations $x_i^2 = x_i$ $(i = q, \ldots, v)$, we find that we can interpret the linear terms of γ as quadratic ones. Therefore, we can consider γ as a homogeneous quadratic polynomial. We store the coefficients of γ according to the graded-reverse-lexicographic order. Therefore we get $\gamma(v_1, \ldots, v_v) = \gamma_{1,1}v_1^2 + \gamma_{2,1} \cdot v_2 \cdot v_1 + \gamma_{2,2} \cdot v_2 \cdot v_2 + \cdots + \gamma_{v,v} \cdot v_v \cdot v_v$. The γ part of $\mathcal{F}(X)$ stores the $v(v+1)/2$ coefficients of $\gamma(v_1, \ldots, v_v)$ as a sequence

$$[\gamma_{1,1}, \gamma_{2,1}, \gamma_{2,2}, \gamma_{3,1}, \ldots, \gamma_{v,v})].$$

Note that each of the $\gamma_{i,j}$ is a vector of n bits.

Example (central map of degree 17): The quadratic monomials in a degree-17 central map are $\{X^2 = X^{2^0+2^0}, X^3 = X^{2^0+2^1}, X^4 = X^{2^1+2^1}, X^5 = X^{2^2+2^0}, X^6 = X^{2^2+2^1}, X^8 = X^{2^2+2^2}, X^9 = X^{2^3+2^0}, X^{10} = X^{2^3+2^1}, X^{12} = X^{2^3+2^2}, X^{16} = X^{2^3+2^3}, X^{17} = X^{2^4+2^0}\}$. These are divided into the sets $A = \{X^2, X^4, X^8, X^{16}\}$ and $B = \{X^3, X^5, X^6, X^9, X^{10}, X^{12}, X^{17}\}$. We denote the coefficient of $X^{q^i+q^j}$ by $\alpha_{i,j}$.
The set of linear monomials is given by $\{X, X^2, X^4, X^8, X^{16}\}$. Note that the monomials X^2, X^4, X^8 and X^{16} are also contained in the set A of quadratic monomials. That is why we do not need constant terms in the linear functions β_i. We denote the linear function multiplied to X^{q^i} by β_i and the coefficient of v_j in β_i by $\beta_{i,j}$.
Finally, we have to append the coefficients of the map γ.

The coefficients of $\mathcal{F}(X)$ are therefore stored in the order

$$[\ \underbrace{\underbrace{\alpha_{0,0},\ \alpha_{1,1},\ \alpha_{2,2},\ \alpha_{3,3}}_{setA}\ ,\ \underbrace{\alpha_{0,1},\ \alpha_{0,2},\ \alpha_{1,2},\ \alpha_{0,3},\ \alpha_{1,3},\ \alpha_{2,3},\ \alpha_{0,4}}_{setB}}_{\text{coefficients } \alpha_{i,j} \text{ of quadratic terms}}\ ,$$

$$\underbrace{\beta_{0,1},\ \ldots,\ \beta_{0,v},\beta_{1,1}\ ,\ldots,\ \beta_{4,v}}_{\text{coefficients of the linear maps } \beta_i}\ ,\quad \underbrace{\gamma_{1,1},\ \gamma_{2,1},\ \gamma_{2,2},\ \ldots,\ \gamma_{v,v}}_{\text{coefficients of the quadratic map } \gamma}\ ].$$

2.4 The field isomorphism ϕ

We use the field of 256 elements, $\mathbb{F}_{256}$, each element occupying 1 byte, as a basic operating unit. The larger extension fields are further extended from $\mathbb{F}_{256}$.

We first describe here the field isomorphism $\phi_{256} : \mathbb{F}_2^8 \to \mathbb{F}_{256}$ and then describe the isomorphism ϕ mapping vectors over $\mathbb{F}_2$ to $\mathbb{E}$.

2.4.1 The isomorphism $\phi_{256} : \mathbb{F}_2^8 \to \mathbb{F}_{256}$

We use the *tower field* representation of $\mathbb{F}_{256}$ which considers an element in $\mathbb{F}_{256}$ as a linear polynomial over $\mathbb{F}_{16}$. Elements of GF(16) are viewed as linear polynomials over GF(4) and so on. The sequence of tower fields from which we build $\mathbb{F}_{256}$ looks like

$$\begin{aligned} \mathbb{F}_4 &:= \mathbb{F}_2[e_1]/(e_1^2 + e_1 + 1), \\ \mathbb{F}_{16} &:= \mathbb{F}_4[e_2]/(e_2^2 + e_2 + e_1), \\ \mathbb{F}_{256} &:= \mathbb{F}_{16}[e_3]/(e_3^2 + e_3 + e_2e_1)\ . \end{aligned}$$

We use $\mathbf{b}_{256} = (1, e_1, e_2, e_1e_2, e_3, e_1e_3, e_2e_3, e_1e_2e_3) \in \mathbb{F}_{256}^8$ as a basis for $\mathbb{F}_{256}$. Such, the $\mathbb{F}_{256}$-element encoded as `0x2` is e_1, `0x4` is e_2, `0x8` is e_1e_2, `0x16` is e_3 etc., and numbers up to `0xff` are their combinations, for example `0x1d` $= e_3 + e_1e_2 + e_2 + 1$.

The field isomorphism ϕ_{256} maps a vector $\mathbf{v} = (v_1, \ldots, v_8) \in \mathbb{F}_2^8$ to the $\mathbb{F}_{256}$ element $v_1 + v_2 \cdot e_1 + \cdots + v_8 \cdot e_1e_2e_3$.

2.4.2 The isomorphism $\phi : \mathbb{F}_2^n \to \mathbb{E}$

A field element in $\mathbb{F}_{2^{184}}$ is represented as a degree 22 polynomial in $\mathbb{F}_{256}[X]$. We define

$$\mathbb{F}_{2^{184}} := \mathbb{F}_{256}[X]/X^{23} + X^3 + X + \texttt{0x2}\ .$$

The isomorphism $\phi : \mathbb{F}_2^{184} \to \mathbb{F}_{2^{184}}$ is defined as

$$\begin{aligned} \phi : \mathbf{a} =& (a_1, \ldots, a_{184}) \in \mathbb{F}_{2^{184}} \mapsto \\ & (b_0, \ldots, b_{22}) = (\phi_{256}(a_1, \ldots, a_8), \ldots, \phi_{256}(a_{177}, \ldots, a_{184})) \in \mathbb{F}_{256}^{22} \mapsto \\ & b_0 + b_1 \cdot X + b_2 \cdot X^2 \ldots + b_{22} \cdot X^{22} \in \mathbb{F}_{256}[X]\ . \end{aligned}$$

The isomorphisms for the other extension fields used for Gui differ from the above construction only by the use of different irreducible polynomials. We define

$$\begin{aligned} \mathbb{F}_{2^{184}} &:= \mathbb{F}_{256}[X]/X^{23} + X^3 + X + \texttt{0x2}, \\ \mathbb{F}_{2^{312}} &:= \mathbb{F}_{256}[X]/X^{39} + X^2 + X + \texttt{0x2}, \\ \mathbb{F}_{2^{448}} &:= \mathbb{F}_{256}[X]/X^{56} + \texttt{0x2} \cdot X^3 + X + \texttt{0x10}. \end{aligned} \tag{2}$$

3 Implementation Details

In this section we present the details of our implementation of the Gui signature scheme.

3.1 Arithmetic Over Finite Fields

The first step in our implementation of the Gui signature scheme is to provide efficient arithmetic operations over the large binary fields in use. To do this, we use a set of new processor instructions for carry-less multiplication: PCLMULQDQ [18].
The instruction set PCLMULQDQ allows the efficient multiplication of two degree 64 polynomials over GF(2), resulting in a polynomial of degree 128. The PCLMULQDQ instructions are available on most new processors of Intel and AMD. Performance data of PCLMULQDQ can be found in Table 1.

Processor type	Latency (cycles)	Throughput (cycles/multiplication)
Intel Sandy Bridge	14	8
Ivy Bridge	14	8
Haswell	7	2
Skylake	7	1
AMD Bulldozer	12	7
Piledriver	12	7
Steamroller	11	7

Table 1: Performance of PCLMULQDQ on different platforms [8]

In the case of Gui, we represent an element of the field $\mathbb{E}$ as a polynomial over GF(2) of degree 184, 312 or 448. These polynomials can be divided into 3–7 polynomials of degree 64, which then can be used as input values for PCMULQDQ. A multiplication over the large field $\mathbb{E}$ is divided into two phases, namely a multiplication and a reduction phase.

In the *multiplication phase*, the multiplication of two 184-bit polynomials can be performed by 6 calls of PCLMULQDQ. With the help of the Karatsuba algorithm, we can avoid 3 calls of PCLMULQDQ and therefore its long latency

(see Table 1).
To square an element of $\mathbb{E}$, we need only 3 calls of PCLMULQDQ since we are operating over a field of characteristic 2.

The *reduction phase* of the field multiplication heavily depends on the representation of the extension fields. The baseline for ths step is 9 calls of PCLMULQDQ, since, after the multiplication phase, the degree of the polynomial will be greater than $5 \cdot 64$.
The irreducible polynomials used to define the extension fields (see equation (2)) are chosen to contain only few terms of low degree. With few terms in the irreducible polynomials, we can replace the use of PCLMULQDQ by a few logic shifts and XOR instructions.

Since, regardless of the input, the same operations are performed, our implementation provides time-constant multiplication for preventing side channel leakage. The same strategy is also applied to the calculation of multiplicative inverses. For example, for the sake of time-constant arithmetics, the inverse of an element $x \in \mathrm{GF}(2^{184})$ is calculated by raising x to $x^{2^{184}-2}$ instead of the faster extended Euclidean algorithm.

3.2 Inverting the HFEv- Core

In this section we describe how we can perform the inversion of the central HFEv- equation $\mathcal{F}_V(\hat{Y}) = X$ efficiently. To invert the central HFEv- equation, we have to run the Berlekamp or Cantor-Zassenhaus algorithms to find the roots of the polynomial $\mathcal{F}_V(\hat{Y}) - X$. In order to speed up the computations, we restrict to polynomials $\mathcal{F}_V(\hat{Y}) - X$ having a unique solution (see Algorithms `GuiSign` and `GuiSign`*). Therefore, we only have to perform the first step of the algorithm, i.e. the computation of

$$\gcd(\mathcal{F}_V(\hat{Y}) - X, \hat{Y}^{2^n} - \hat{Y}). \tag{3}$$

In case that this gcd is not a linear polynomial, we choose another value for the random seed r and try again. We have

$$\begin{aligned} & \gcd(\mathcal{F}_V(\hat{Y}) - X, \hat{Y}^{2^n} - \hat{Y}) \\ = \quad & \gcd(\mathcal{F}_V(\hat{Y}) - X, \prod_{i \in \mathbb{F}_{2^n}} (\hat{Y} - i)) = \prod_{i \in \mathbb{F}_{2^n} : \mathcal{F}_V(i) = X} (\hat{Y} - i). \end{aligned}$$

Therefore the most costly step in generating a signature consists in computing $\gcd(\mathcal{F}_V(\hat{Y}) - X, \hat{Y}^{2^n} - \hat{Y})$. The number of roots of $\mathcal{F}_V(\hat{Y}) - X$ (as well as the only solution when that happens) can obviously be read off from the result.

Probability of a Unique Root. Every time we choose the values of Minus equations and Vinegar variables (respectively, when we pick a salt), we basically pick a random central equation $\mathcal{F}_V(\hat{Y}) - X = 0$. The probability of this

equation having a unique solution is about $1/e$. Therefore, in order to invert the HFEv- central equation k times successfully, we have to perform the gcd computation about e^k times.
The repeated computation of the gcd is probably the most detectable side channel leakage of our scheme. However, there are no known side channel attacks on big field schemes or HFEv- which use the information that one particular equation in the big field has no, respectively two or more solutions.

How Do We Optimize the Computation of the GCD? The most costly step in the computation of the gcd is the division of the extreme high power polynomial $\hat{Y}^{2^n} - \hat{Y} \bmod \mathcal{F}_V(\hat{Y})$. A naive long division is unacceptable for this purpose due to its slow reduction phase. Instead of this, we choose to recursively raise the lower degree polynomial $\hat{Y}^{2^m}$ to the power of 2.

$$\begin{aligned} & \left(\hat{Y}^{2^m} \bmod \mathcal{F}_V(\hat{Y})\right)^2 \\ = & \left(\sum_{i \le D} b_i \hat{Y}^i\right)^2 \bmod \mathcal{F}_V(\hat{Y}) = \left(\sum_{i \le D} b_i^2 \hat{Y}^{2i}\right) \bmod \mathcal{F}_V(\hat{Y}) \end{aligned}$$

By multiplying $\hat{Y}$ to the naive relation

$$\hat{Y}^D = \sum_{0 \le i \le j, 2^i + 2^j < D} \alpha_{ij} \hat{Y}^{2^i + 2^j},$$

we can prepare a table for $\hat{Y}^{2^i} \bmod \mathcal{F}_V(\hat{Y})$ first. The remaining computation of the raising process consists in squaring all the coefficients b_i in $\hat{Y}^{2^m} \bmod \mathcal{F}_V(\hat{Y})$ and multiply them to the $\hat{Y}^{2i}$'s in the table.
Although the starting relation

$$\mathcal{F}_V(\hat{Y}) = \hat{Y}^D + \sum_{0 \le i \le j,\ 2^i + 2^j < D} \alpha_{ij} \hat{Y}^{2^i + 2^j}$$

is a sparse polynomial, the polynomials become dense quickly in the course of the raising process. However, the number of terms in the polynomials is restricted by D because of mod $\mathcal{F}_V(\hat{Y})$. We expect the number of terms to be in average D during the computation.

We implemented the simplified Cantor-Zassenhaus algorithm in such a way that it takes, independently of the input, always the same number of iterations in the main gcd loop and the same number of operations in the big field. Therefore the algorithm runs, independently from the input, at constant time.

The number of field multiplications needed to compute the $\hat{Y}^{2^i}$ table is $\mathcal{O}(2 \cdot D^2)$. To raise $\hat{Y}^{2^m}$ to $\hat{Y}^{2^n}$, we need $\mathcal{O}((n-m) \cdot D)$ squarings and $\mathcal{O}((n-m) \cdot D^2)$

multiplications.
It is possible to reduce the number of computations needed for computing $\hat{Y}^{2^m}$ further by using a higher degree $\hat{Y}^i$ table. For example, if one raises $\hat{Y}^{2^m}$ to $\hat{Y}^{2^{4m}}$ in one step, one only needs $\mathcal{O}((n-m)\cdot D)$ squarings and $\mathcal{O}((n-m)\cdot 2\cdot D^2)$ multiplications. However, the computational effort of preparing the $\hat{Y}^{2^i}$ table increases.

4 Performance Analysis

4.1 Key and Signature Sizes

The following table shows the key and signature sizes of our proposed Gui-instances.

	parameters (n, D, a, v, k)	public key size (kB)	private key size (kB)	signature size (bit)[1]
Gui-184	$(184, 33, 16, 16, 2)$	416.3	19.1	360
Gui-312	$(312, 129, 24, 20, 2)$	1,955.1	59.3	504
Gui-448	$(448, 513, 32, 28, 2)$	5,789.2	155.9	664

[1] including 128 bit salt

Table 2: Key and Signature sizes of the proposed Gui instances

4.2 Performance on the NIST Reference Platform

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

scheme	parameters (n, D, a, v, k)		key generation	signature generation	signature verification
Gui-184	$(184, 33, 16, 16, 2)$	cycles	2,408M	1,910M	152k
		time (ms)	213	10.4	0.051
		memory	3.5MB	3.4MB	3.3MB
Gui-312	$(312, 129, 24, 20, 2)$	cycles	43,817M	25,436M	846k
		time (ms)	13,227	7,707	0.256
		memory	5.4MB	3.8MB	5.0MB
Gui-448	$(448, 513, 32, 28, 2)$	cycles	239,502M	872,949M	1,787k
		time (ms)	71,485	264,530	0.542
		memory	17.7MB	10.7MB	8.7MB

Table 3: Performance of Gui on the NIST Reference Platform (Linux/Skylake)

4.3 Performance on Other Platforms

Processor: Intel(R) Xeon(R) CPU E3-1225 v5 @ 3.30GHz (Skylake)
Clock Speed: 3.30GHz
Memory: 64GB (4x16) ECC DIMM DDR4 Synchronous 2,133 MHz (0.5 ns)
Operating System: Linux 4.8.5, GCC compiler version 6.4
Use of PCLMULQDQ instructions

scheme	parameters (n, D, a, v, k)		key generation	signature generation	signature verification
Gui-184	$(184, 33, 16, 16, 2)$	cycles	704M	34M	169k
		time (ms)	213	10.4	0.051
		memory	3.5MB	3.4MB	3.3MB
Gui-312	$(312, 129, 24, 20, 2)$	cycles	4,790M	1,757M	595k
		time (ms)	1,452	532	0.181
		memory	5.4MB	3.6MB	5.0MB
Gui-448	$(448, 513, 32, 28, 2)$	cycles	32,247M	86,086M	3,385k
		time (ms)	9,772	26,086	1.025
		memory	9.2MB	10.7MB	8.7MB

Table 4: Performance of Gui on Linux/Skylake (PCLMULQDQ)

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

scheme	parameters (n, D, a, v, k)		key generation	signature generation	signature verification
Gui-184	$(184, 33, 16, 16, 2)$	cycles	721M	34M	141k
		time (ms)	343	16.3	0.067
		memory	3.7MB	3.2MB	3.2MB
Gui-312	$(312, 129, 24, 20, 2)$	cycles	4,955M	1,815M	371k
		time (ms)	2,360	864	0.018
		memory	5.3MB	3.5MB	4.8MB
Gui-448	$(448, 513, 32, 28, 2)$	cycles	30,025M	88,528M	3,307k
		time (ms)	71,485	42,156	1.575
		memory	17.5MB	10.7MB	8.6MB

Table 5: Performance of Gui on Linux/Broadwell (PCLMULQDQ)

4.4 Note about the measurements

Turboboost is disabled on our platforms. The main compilation flags are `gcc -O3 -std=c99 -Wall -Wextra (-mclmul)`. The used memory is measured during an actual run using `/usr/bin/time -f "%M"` (average of 10 runs). For key generation, signatures and verification we take the average of 10 runs.

4.5 Trends as the number n of variables increases

Key Generation: Like other large field multivariate schemes, HFEv-/Gui uses interpolation to generate the keys, which takes about n^2 times the time needed to evaluate the central map once. This is often said to be an $O(n^6)$ operation. In practice, the HFE polynomial has $O((\log_q D)^2)$ terms, each of which takes $O(\log_q D)$ square-and multiplies in the big field, so the complexity is closer to $O(n^4 \log_q D^3)$ in our range.

Verification/Public Map: This is a straightforward MQ evaluation and is $O(n^3)$,

Signature/Secret Map: The simplified Cantor-Zassenhaus algorithm takes $O(nD)$ big field multiplications, but is in our implementation in practice closer to $O(nD^2)$. Similarly, the big field multiplication is $O(n \log n)$ but is in practice is closer to $O(n^2)$, so in our range the time used increase more like $O(n^3 D^2)$.

5 Expected Security Strength

In the NIST call for proposals [11], the security of a scheme is measured in the number of classical or quantum gates that an attack against the scheme requires. Hereby, the number of quantum gates contains a factor `MAXDEPTH`, which, according to [11], can take the values 2^{40}, 2^{64}, or 2^{96}. According to this, the proposed 6 NIST security categories are defined as shown in Table 6.

category	$\log_2$ classical gates	$\log_2$ (logical) quantum gates
I	143	130 / **106** / 74
II	146	
III	207	193 / **169** / 137
IV	210	
V	272	258 / **234** / 202
VI	274	

Table 6: NIST security categories

In this proposal, we use a value of `MAXDEPTH` of 2^{64}. This seems plausible to us since, in a quantum setting, the best known attack against our scheme is the quantum brute force attack, for which we assume `MAXDEPTH` to be more or less in the middle of the given extremas. This choice results in the values printed in **bold** in Table 6.

5.1 General Remarks

The Gui signature scheme as described in Section 1.6 of this proposal fulfills the requirements of the EUF-CMA security model (existential unforeability under chosen message attacks). The parameters of the scheme (in particular the length of the random salt) are chosen in a way that up to 2^{64} messages can be signed with one key pair. The scheme can generate signatures for messages of arbitrary length (as long as the underlying hash function can process them).

5.2 Practical Security

In this section we study the practical security of the proposed instances of the Gui signature scheme. Since the security of Gui can not be reduced directly to a hard mathematical problem such as the $\mathcal{MQ}$-Problem, we choose the parameters of Gui in a way that the complexities of all known attacks against Gui are beyond the required levels of security. We note that, despite of the altered signature generation process, all of the known attacks against Gui are attacks against the underlying HFEv- scheme. These include (see Section 6):

- brute force attacks (see Section 6.2)
- direct algebraic attacks (see Section 6.3)

- rank attacks of the Kipnis-Shamir type (see Section 6.4) and
- the distinguishing attack (see Section 6.5).

While the first two of these attacks are signature forgery attacks, which have to be performed for every message separately, the Kipnis-Shamir attack and the distinguishing attack are key recovery attacks. After having recovered the HFEv- private key using one of these attacks, the attacker can generate signatures for arbitrary messages in the same way as a legitimate user. Further note that, due to the special signature generation process of Gui, the attacker has, in order to forge a Gui signature, to run the HFEv- signature forgery attacks k times.

The following table shows the complexity of these known attacks against our Gui instances. In each cell, the first number shows the number of classical gates required to perform the attack, while the second number shows the required number of quantum gates. In each row, the number printed in **bold** shows the complexity of the best attack against the scheme.

	parameters	$\log_2$(#gates)			
	(n, D, a, v, k)	direct[1]	brute force	MinRank[1]	distinguisher
Gui-184	$(184, 33, 16, 16, 2)$	**156.8**	171.9	323.6	246.4
		156.8	**108.2**	323.6	191.4
Gui-312	$(312, 129, 24, 20, 2)$	**221.6**	292.0	480.5	370.7
		221.6	**170.5**	480.5	280.3
Gui-448	$(448, 513, 32, 28, 2)$	**293.3**	420.1	665.2	509.9
		293.3	**236.1**	665.2	381.9

[1] As our analysis (see Section 6) shows, there is no difference between the number of classical and quantum gates for direct and rank attacks.

Table 7: Estimated attack complexities against the proposed Gui instances

Based on the above security evaluation (and the results of Section 5.3 below) , we propose Gui-184 for the security categories I and II (see Table 4). Gui-312 meets the requirements of security categories III and IV, whereas Gui-448 is suitable for the security Categories V and VI.

5.3 Security against collision attacks

Additionally to the attacks against the Gui scheme itself, we also have to study the security of our schemes under collision attacks against the underlying hash functions. As already mentioned in Section 1.8.1, we use

- SHA256 for Gui-184
- SHA384 for Gui-312 and

- SHA512 for Gui-448

as hash functions underlying the proposed Gui instances. For all of the proposed schemes, we use $k = 2$ as the repetition factor of our scheme. Therefore, we obtain for the parameter ℓ used in Algorithms 2, 4, 6 and 8

$$\ell = \lceil k \cdot (n-a)/|\mathcal{H}| \rceil = 2,$$

which means that the vector $\tilde{\mathbf{h}}$ used in these algorithms is given by

$$\tilde{\mathbf{h}} = \mathcal{H}(d)||\mathcal{H}(\mathcal{H}(d)) \quad (\text{rsp. } \mathcal{H}(\mathcal{H}(d)||r)||\mathcal{H}(\mathcal{H}(\mathcal{H}(d)||r)).$$

The first hash value used in the signature generation process of Gui, $\mathbf{d}_1$ consists of the first $(n-a)$ bits of $\tilde{\mathbf{h}}$, whereas $\mathbf{d}_2 = \tilde{\mathbf{h}}_{n-a+1}, \ldots, \tilde{\mathbf{h}}_{2\cdot(n-a)}$.
In any case, all the bits of $\mathcal{H}(d)$ are contained either in $\mathbf{d}_1$ or $\mathbf{d}_2$, which means that a collision attack against our scheme is at least as hard as a collision attack against the hash function $\mathcal{H}$. This justifies the classification of our Gui instances into the corresponding security categories.

5.4 Side Channel Resistance

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

6 Analysis of Known Attacks

Despite of the modified signature generation process, the security of Gui is based on that of the HFEv- scheme and all known attacks against Gui are attacks against HFEv-. These include

- collision attacks against the hash function (Section 6.1)
- brute force attacks (Section 6.2)
- direct attacks (Section 6.3)
- rank attacks of the Kipnis-Shamir type (Section 6.4)
- the distinguishing attack of Perlner et al. (Section 6.5).

Brute force and direct attacks are signature forgery attacks, which means that they have to be performed for each message separately. Furthermore, due to the special signature generation process of Gui, these attacks have to be performed k times in order to forge a Gui signature. On the other hand, the rank attack of Kipnis and Shamir and the distinguishing attack are key recovery attacks. After having recovered the HFEv- private key using one of these attacks, the adversary can sign messages in the same way as a legitimate user.

6.1 Collision attacks against the hash function

Since the Gui signature scheme follows the Hash then Sign approach, we have to choose the parameters of Gui in such a way that collision attacks against the underlying hash function are infeasible. However, due to the specially designed signature generation process of Gui, this does not have a direct influence on the number of equations in the public system. Since a Gui signature depends on the k hash values $\mathbf{d}_1, \ldots, \mathbf{d}_k \in \mathbb{F}^{n-a}$, the resulting effective length of the hash value is $k \cdot (n-a) \cdot \log_2 q$ bit. Therefore, the complexity of a collision attack against our scheme is about

$$\text{Complexity}_{\text{collision}} = 2^{k \cdot (n-a)}.$$

By choosing the repetition factor k in an appropriate way, it is therefore easy to prevent collision attacks (even for a relatively small number $(n-a)$ of equations).
An analysis of the security of our proposed Gui instances against collision attacks can be found in Section 5.3.

6.2 Brute Force Attacks

Since the public system of Gui is defined over the field GF(2) with two elements, the parameters of the scheme have to be chosen in a way that prevents brute force attacks against binary $\mathcal{MQ}$-systems. In the classical world, we have to

mention here the Gray Code enumeration of [3]. In order to solve a public HFEv- system using this technique, one first fixes $v + a$ variables to get a determined system. The resulting system of $n - a$ equations in $n - a$ variables can then be evaluated for every possible input using $2^{n-a+2} \cdot \log_2(n-a)$ bit operations. In order to forge a Gui signature, we have to perform this step k times. Therefore, the complexity of this attack can be estimated as

$$\text{Complexity}_{\text{brute force; classical}} = k \cdot 2^{n-a+2} \cdot \log_2(n-a)$$

bit operations.
In the quantum world, brute force attacks can be sped up using Grover's algorithm. As shown in [17], we can find the solution of a binary $\mathcal{MQ}$-system of $n - a$ equations in $n - a$ variables using $2^{(n-a)/2} \cdot 2 \cdot (n-a)^3$ quantum bit operations. Again, in order to forge a Gui signature, an attacker has to perform this step k times. Therefore, we can estimate the complexity of a quantum brute force attack against our scheme as

$$\text{Complexity}_{\text{brute force; quantum}} = k \cdot 2^{(n-a)/2} \cdot 2 \cdot (n-a)^3$$

(quantum) bit operations.

6.3 Direct Attacks

Similar to the brute force attacks (see previous section), a direct attack considers the public equation $\mathcal{P}(\mathbf{z}) = \mathbf{w}$ as an instance of the $\mathcal{MQ}$-Problem. Since the public system of HFEv- is an underdetermined system (more variables than equations), the most efficient way to solve this equation is to fix $a + v$ variables to create a determined system before applying an algorithm like XL or a Gröbner Basis technique such as F_4 or F_5 [6]. It can be expected that the resulting determined system has exactly one solution. In some cases, one obtains even better results when guessing additional variables before solving the system (hybrid approach) [1].

Experiments [10, 7] have shown that the public systems of HFEv- can be solved significantly faster than random systems. The reason for this is that these systems have a significantly lower degree of regularity. In [5] it was shown that the degree of regularity of an HFEv- system is upper bounded by

$$\mathrm{d_{reg}} \leq \begin{cases} \frac{(q-1)\cdot(r+a+v-1)}{2} + 2 & q \text{ even and } r+a \text{ odd,} \\ \frac{(q-1)\cdot(r+a+v)}{2} + 2 & \text{otherwise.} \end{cases} \tag{4}$$

with $r = \lfloor \log_q(D-1) \rfloor + 1$. Since the upper bound on the degree of regularity given by equation (4) does not really help to estimate the complexity of direct attacks against HFEv- schemes in practice, we follow here the analysis of [14]. In [14], Petzoldt derived from experiments the following lower bound for the degree of regularity of an HFEv- system

$$d_{\mathrm{reg}} = \lfloor \frac{a+r+v+7}{3} \rfloor \tag{5}$$

Furthermore, as shown in [14], the hybrid approach does not help to speed up direct attacks against HFEv- schemes. In our security analysis (see previous section), we therefore estimate the complexity of a direct attack against an HFEv-(n, D, a, v) instance as

$$\text{Complexity}_{\text{direct attack}} = 2 \cdot k \cdot 3 \cdot \binom{n-a}{d_{\text{reg}}}^2 \cdot \binom{n-a}{2}$$

bit operations, where d_{reg} is given by formula (5).
Since there is no guessing step in the attack, we can not reduce its complexity by the use of Grover's algorithm.

6.4 Rank attacks of the Kipnis Shamir type

In [9], Kipnis and Shamir proposed a rank attack against the HFE cryptosystem. The key idea of this attack is to consider the public and private maps of HFE as univariate polynomial maps over the extension field. Due to the special structure of the HFE central map, the rank of the corresponding matrix is limited by $r = \lfloor \log_2(D-1) \rfloor + 1$. It is therefore possible to recover the affine transformation $\mathcal{S}$ by solving an instance of the MinRank problem.

In [2], Bouillaget et al. improved this attack by showing that the map $\mathcal{S}$ can be found by computing a Gröbner Basis over the base field GF(2) (Minors Modelling). By doing so, they could speed up the attack of Kipnis and Shamir significantly. The complexity of the MinRank attack against HFE using the Minors Modelling approach can be estimated as

$$\text{Complexity}_{\text{MinRank; HFE}} = \binom{n+r}{r}^{\omega},$$

where $r = \lfloor \log_2(D-1) \rfloor + 1$ is the rank of the matrix corresponding to the central map and $2 < \omega \leq 3$ is the linear algebra factor. In our security analysis (see Secton 5.2), we choose the value of ω to be 2.3.
In the case of HFEv-, the rank of the matrix is given by $r + a + v$ [2]. Therefore, we can estimate the complexity of our attack by

$$\text{Complexity}_{\text{KS attack; HFEv}-} = \begin{cases} \binom{n+r+a+v}{r+a+v}^{2.3} & r+a+v \text{ even} \\ \binom{n+r+a+v-1}{r+a+v-1}^{2.3} & r+a+v \text{ odd} \end{cases}.$$

Since the quadratic systems to be solved during this attack are highly overdetermined, the attack can not be sped up with the help of Grover's algorithm.

[2] Since we work over fields of even chatacteristic, the rank of the matrix corresponding to the central map $\mathcal{F}$ is always even. Therefore, in the case of $r + a + v$ being odd, the rank of this matrix is given by $r + a + v - 1$.

6.5 The Distinguishing attack of Perlner et al.

The distinguishing attack of Perlner et al. [13] uses the fact that the behavior of a direct attack depends on the number of vinegar variables in the HFEv- system. By using this fact, it is possible to remove the vinegar variables one by one from the system. The resulting HFE- system can then be solved much easier than the original system. The most costly step in the attack is hereby to remove the first vinegar variable, i.e. the reduction of an HFEv-(n, D, a, v) to an HFEv-$(n, D, a, v-1)$ scheme. The complexity of this first step can be estimated as

$$\text{Complexity}_{\text{Distinguisher; classical}} = 2^{n-k} \cdot 3 \cdot \binom{n+v-k}{d_{\text{reg}}}^2 \cdot \binom{n+v-k}{2},$$

where

- $n + v - k$ is the maximal number of variables for which a direct attack against the projected systems HFEv-(n, D, a, v) and HFEv-$(n, D, a, v-1)$ behaves differently; $n+v-k$ is given as the maximal number n' for which the degree of regularity of a direct attack against a projected random system of $(n - a)$ quadratic equations in n' variables is below d_{reg}.
- d_{reg} is the degree of regularity of a direct attack against the unprojected HFEv-(n, D, a, v) system; according to [14], d_{reg} can be estimated by

$$d_{\text{reg}} = \lceil \frac{r + a + v + 7}{3} \rceil.$$

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

$$\text{Complexity}_{\text{Distinguisher; quantum}} = 2^{(n-k)/2} \cdot 3 \cdot \binom{n+v-k}{d_{\text{reg}}}^2 \cdot \binom{n+v-k}{2}.$$

6.6 Differential attacks

Differential attacks against the HFEv- scheme were intensively studied in [4]. In this paper, the authors proved that HFEv- has no differential symmetries or invariants which could be used for differential attacks.

7 Advantages and Limitations

The main advantages of the Gui signature scheme are

- **Very short signatures**. The signatures produced by the Gui signature scheme are of size about two times the corresponding security level. Therefore, Gui produces the shortest signatures of all existing digital signature schemes (both classical and post-quantum).

- **Security**. Though there exists no formal security proof which connects the security of the Gui signature scheme to a hard mathematical problem such as MQ, we are quite confident about the security of our scheme. The Gui signature scheme is based on the HFEv- signature scheme, which is one of the best known and most analyzed multivariate schemes. The only recent advance in the cryptanalysis of HFEv- like schemes is the Minors modelling of the MinRank attack [2] from 2013. However, as shown in [15], this attack can be easily prevented by increasing the numbers of minus equations and vinegar variables. Moreover, while the behaviour of direct attacks against Gui/HFEv-had long been a mystery, this problem could be solved by the works of [5, 15, 14].
  In general we can say that, in the case of the Gui signature scheme, the experimental data follow closely the theoretical complexity estimations of the known attacks. This is fundamentally different than for many other cryptographic schemes, e.g. lattice based constructions, and gives us additional confidence in the security of Gui.

- **Modest computational requirements**. Since Gui only requires simple linear algebra operations over a small finite field, it can be efficiently implemented on low cost devices, without the need of a cryptographic coprocessor [15].

- **Efficiency**. Though Gui is not one of the fastest multivariate schemes, its performance is highly comparable with that of RSA and ECC (see [15] and Section 4.2). Especially for high levels of security, the parameters of Gui and therefore the running time do not increase as drastically as in the case of RSA. We further mention here that, in the last years, the performance of schemes like Gui has improved dramatically due to new processor instructions such as `PCLMULQDQ`.

On the other hand, the main disadvantage of Gui is its **Large Public Key Size**. The public key size of Gui lies, for the parameter sets recommended in this proposal, in the range of 400 kB to 5MB and is therefore much larger than that of many classical signature schemes such as RSA and DSA and e.g. lattice based signature schemes.
On the other hand, the private key of Gui is much smaller than the public key, which allows to store the private key on small devices such as smartcards.

References

[1] L. Bettale, J.C. Faugére, L. Perret: Hybrid approach for solving multivariate systems over finite fields. J. Math. Cryptol. 3, pp. 177 - 197 (2009).

[2] L. Bettale, J.C. Faugére, L. Perret: Cryptanalysis of HFE, multi-HFE and variants for odd and even characteristic. Designs Codes and Cryptography 69 (2013), pp. 1 - 52.

[3] C. Bouillaguet, H.-C. Chen, C.-M. Cheng, T. Chou, R. Niederhagen, A. Shamir, B.-Y. Yang: Fast exhaustive search for polynomial systems in F2. CHES 2010, LNCS vol. 6225, pp. 203 - 218. Springer, 2010.

[4] R. Cartor, R. Gipson, D. Smith-Tone, J. Vates: On the Differential Security of the HFEv- Signature Primitive. PQCrypto 2016, LNCS vol. 9606, pp. 162 - 181. Springer, 2016.

[5] J. Ding, B.Y. Yang: Degree of Regularity for HFEv and HFEv-. PQCrypto 2013, LNCS vol. 7932, pp. 52 - 66. Springer, 2013.

[6] J.C. Faugére: A new efficient algorithm for computing Gröbner bases (F4). J. Pure Appl. Algebra 139, pp. 61 - 88 (1999).

[7] J.C. Faugére: Algebraic cryptanalysis of Hidden Field Equations (HFE) using Gröbner bases. CRYPTO 2003, LNCS vol. 2729, pp. 44 - 60. Springer, 2003.

[8] A. Fog: Instruction tables: Lists of instruction latencies, throughputs and microoperation breakdowns for Intel, AMD and VIA CPUs, 7 December 2014. `http:// www.agner.org/optimize/`

[9] A. Kipnis, A. Shamir: Cryptanalysis of the HFE public key cryptosystem by Relinearization. CRYPTO 1999, LNCS vol. 1666, pp. 19 - 30. Springer, 1999.

[10] M.S.E. Mohamed, J. Ding, J. Buchmann: Towards algebraic cryptanalysis of HFE challenge 2. ISA 2011. CCIS vol. 200, pp. 123 - 131. Springer, 2011.

[11] NIST: Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/submission-requirements`

[12] J. Patarin, N.T. Courtois, L. Goubin: QUARTZ, 128-bit long digital signatures. CT-RSA 2001, LNCS, vol. 2020, pp. 282 - 297. Springer, 2001.

[13] R. Perlner, A. Petzoldt, D. Smith - Tone: Improved Cryptanalysis of HFEv- via Projection. Submitted to PQCrypto 2018. Available as IACR eprint report 2017/1149.

[14] A. Petzoldt: On the Complexity of the Hybrid Approch on HFEv-. IACR eprint report 2017/1135.

[15] A. Petzoldt, M.S. Chen, B.Y. Yang, C. Tao, J. Ding: Design principles for HFEv- based multivariate signature schemes. ASIACRYPT 2015 (Part 1), LNCS vol. 9742 , pp. 311 - 334. Springer, 2015.

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

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

[18] J. Taverne, A. Faz-Hernandez, D.F. Aranha, F. Rodrguez-Henrquez, D. Hankerson, , J. Lopez: Software implementation of binary elliptic curves: impact of the carry-less multiplier on scalar multiplication. CHES 2011. LNCS vol. 6917, pp. 108123. Springer, 2011.

[19] J. Vates, D. Smith-Tone: Key recovery for all parameters of HFE-. PQCrypto 2017, LNCS 10346, pp. 272 -288. Springer, 2017.