# HPPC: Hidden Product of Polynomial Composition

Borja Gomez Rodriguez

borja.gomez@develrox.com

31-05-2023

**Abstract**

The article introduces **HPPC** a new Digital Signature scheme that intends to resist known previous attacks applied to HFE-based schemes like QUARTZ and GeMMS. The idea is to use maximal degree for the central HFE polynomial whereas the trapdoor polynomial has low degree in order to sign messages by finding polynomial roots in an extension field via Berlekamp's algorithm. This work has been submitted to NIST's Post-Quantum Cryptography challenge (PQC) and code is available at Github

# 1 Introduction

This paper introduces HPPC, a new digital signature scheme based on Multivariate Public Key Cryptography (MPKC) . The idea behind this scheme is to hide the multiplication and the composition of Linearised Permutation Polynomials such that the owner can transform the private function $F(X) = Y$ into an easier one $G(X') = Y$ such that recovering $X'$ gives the original value $X$ immediately. The scheme is based on the ideas of HFE [Patb], where other NIST-PQC candidate GeMSS [Cas] was based in similar ideas. However, there are major differences between HPPC and GeMSS, as HPPC uses a big degree private polynomial which makes discovered attacks unfeasible (apparently).

## 1.1 Background

The field of Multivariate Public Key Cryptography counts with a taxonomy that categorizes schemes into families: BigField, MediumField, Stepwise, Oil-Vinegar [WP]. Schemes based on these families play an important role in the development and study of Post-Quantum schemes as there is a need for strengthening key exchange found in the Internet (i.e: SSL/TLS) and for digital signatures. It's theorized that in next decades Quantum computers will be ready to break the schemes that we use today, as they're based in common problems found in commutative cryptography where the security relies in the Discrete Logarithm Problem and Integer Factorization.

With the introduction of the C* scheme by Matsumoto-Imai the field of MPKC started to gain attention [TH]. The C* scheme was broken in the work of Patarin [Pata] by a Differential attack. Attacker gathers plaintext and ciphertext pairs and mounts a linear equation system that recovers the coefficient of the quadratic equations obtained by the Differential. With these equations Patarin demonstrated that plaintext recovery is doable for C*. Other variations were done like the Perturbation modifier.

After this breakthrough HFE [Patb] gained attention, which is a Dembowski-Ostrom private polynomial, that is represented as a quadratic set (system of quadratic equations). HFE has its weakness on decipher stage, where the Degree of the central polynomial $F(X)$ must be small to apply root finding (e.g: Berlekamp's Trace). Kipnis and Shamir published a work [KS] demonstrating that private polynomial computation is feasible solving the Minrank problem by solving a multivariate system of equations using the relinearization technique. This is because the rank of the private polynomial is considered small.

Variations of HFE appeared to protect from these key recovery attacks. (Gui, HFEv-, GeMMs, QUARTZ), that nowadays are considered not secure as it's been proved that are not resistant to recent discoveries [STV] [TV]. In addition, the underlying problems of MPKC have been broadly studied: PoSSo, Minrank [Bus], Isomorphism of Polynomials (IP2) [Pata] .

With that all in mind the design of **HPPC** is similar to previous HFE based schemes, with the difference that the central polynomial $F(X)$ has big degree. In literature the degree of the central polynomial $D$ must be low to apply Berlekamp's for root finding. In HPPC the public key is composed of a central polynomial with big degree. The private key has low $D$ instead in order to apply root finding. These properties make HPPC resistant to well-known attacks at first glance.

# 2 Algorithm Specification

## 2.1 Tensor Algebra and MPKC Schemes

Since the early beginning of the field of Multivariate Cryptography, schemes belonging to the BigField [WP] families have represented the public key equations encoded in symbolic notation or in quadratic forms. The private polynomial is an univariate polynomial over $F_{q^n}$ instead. Notice that every group $G$ admits a group representation using matrices, in this work we use that fact to construct the scheme using the representation of $G$ using tensors to multiply elements in $G$. There are various mathematical facts that gives us a clue on how to represent efficiently any element of the group G by constructing a basis for a vector space V.

### 2.1.1 Representation of a Finite Field

A Finite Field is a mathematical structure having $q^n$ elements where the axioms of a ring and a group are satisfied. To build such algebraic structure we need an irreducible polynomial $f(x)$ of degree $n$ over the prime field $F_q$. Now use the fact that the *Companion Matrix* of $f(x)$ , which we call $C_{f(x)}$, is used to build a basis of the Finite Field $F_{q^n} = Z_q/\langle f(x)\rangle$. Take a polynomial $g(x) = \sum_{i=0}^{n-1} g_i x^i \in F_{q^n}$, represent it as the vector $g = (g_0, \dots, g_{n-1})$. Now build the matrix representation of $g(x)$ as:

$$B = ({C_{f(x)}}^0 g, \dots, {C_{f(x)}}^{n-1} g)$$

### 2.1.2 Multiplication of elements

Then for multiplying any element $s(x)$ by $g(x)$ we compute $B \cdot s = r$ where $s(x) * g(x) = r(x)$ and $r, s$ are the polynomial vector representation over $F_q$. Based in this property we can devise the Tensor representation of multiplication in $F_{q^n}$, which is the principle of representing MPKC schemes using Tensor Algebra.

### 2.1.3 Tensor representation

Let's represent the operation $g(x)^2 = g(x) * g(x)$ in the basis as $B \cdot g = g^2$:

$$B = ({C_{f(x)}}^0 g g_1, \dots, {C_{f(x)}}^{n-1} g g_n)$$

The reader can notice that the dot product of $B$ and the vector $g = (g_1, \dots, g_n)$ give entries $gg_i = (g_1 g_i, \dots, g_n g_i) = (g_i g_1, \dots, g_i g_n)$ as the base field $F_q$ is commutative. These entries are equal to the tensor product representation of $g \otimes g$. We finally conclude that the tensor representation of multiplication of elements $(g(x), s(x))$ in Finite Field $F_{q^n}$ is:

$$[{C_{f(x)}}^0, \dots, {C_{f(x)}}^{n-1}] \cdot (g \otimes s)$$

Where $M = [{C_{f(x)}}^0, \dots, {C_{f(x)}}^{n-1}]$ is the concatenation of powers of the Companion Matrix of the irreducible polynomial $f(x)$ The last step is to adapt this representation for building MPKC schemes. If we set the multiplication to $M \cdot (g \times g)$ we end up having the Frobenius map $F(X) = X^2$ which is linear and not desirable in MPKC for obvious reasons.

Furthermore, by modifying the representation we can turn this map into a more interesting one by using the fact that group of Permutation Linearised polynomials over $F_q$ is isomorphic to the General Linear Group $GL(q, n)$.

This means that any invertible matrix $M_l$ over $F_q$ acts on the vector $g$ equally as its Linearised Permutation Polynomial equivalent over $F_{q^n}$. This is $M_l \cdot g = \phi(l(g(x)))$, where $\phi(x)$ sends the polynomial $l(g(x))$ to its vector representation, concluding that any invertible matrix $L$ is associated with a Linearised Permutation Polynomial $l(x)$ over $F_{q^n}$. Now select invertible matrices $T, S, L_1, L_2 \in F_q^{n \times n}$ and let's build an example of the representation of MPKC BigField public key $P(X)$.

$$P'(x) = T \cdot M \cdot (L_1 \otimes L_2) \cdot (S \otimes S) \cdot (x \otimes x)$$

Which is equal to its equivalent polynomial $P(X) = T \circ (l_1 * l_2) \circ S(X)$ over $F_{q^n}$ as $P'(x) = \phi(P(X))$.

With this representation in mind we can build MPKC schemes using any platform group $G$ that has interesting properties, for enciphering/deciphering data or for digital signature.

### 2.1.4 HFE Tensor Representation

The cryptosystem HFE uses a Dembowski-Ostrom polynomial over $F_{q^n}$ where $q = p^k$ and $deg(F(X)) = D = q^a + q^b$. Values $\alpha, \beta, \delta$ are defined in the extension field $F_{q^n}$

$$F(X) = \sum_{\substack{i,j=0 \\ q^i+q^j \leq D}} \alpha X^{q^i+q^j} + \sum_{i=0}^{d} \beta X^{q^i} + \sum_{i=1}^{n} \delta_i \in F_{q^n}[X]$$

Now any monomial $X^a$ will have exponent equal or less than $D$. For representing HFE in its Tensor public key form we must compute the matrix representation of each Frobenius Map of kind $\mathrm{Frob}(X) = X^{q^i}, 0 \leq i \leq n-1$. Once obtained, represent polynomials $\alpha, \beta, \delta$ as matrices (as seen in previous section). We can rewrite HFE public key $P(X) = T \circ F \circ S(X)$ as:

$$
\begin{aligned}
M &= ({C_{f(x)}}^{0}, \ldots, {C_{f(x)}}^{n-1}) \\
F(x) &= \sum_{\substack{i,j=0 \\ q^i+q^j \leq D}} (M_{\alpha_{i,j}} \cdot (M_{X_{q^i}} \otimes M_{X_{q^j}})) + \sum_{i=0}^{d} M_{\beta_i} \cdot M_{X^{q^i}} + \sum_{i=1}^{n} M_{\delta_i} \\
P(x) &= T \cdot M \cdot F \cdot (S \otimes S) \cdot (x \otimes x)
\end{aligned}
$$

The resulting public key matrix $P(x)$ has size $n \times n^2$ and its bit-size is $\log_2 q^n n^3$. The rows of $P(X)$ are vectors of $n^2$ which form the Quadratic Forms of the quadratic map. This because the Tensor Product is related to Bilinear Forms $B(x, y)$, here the Quadratic Forms are Bilinear of kind $B(x, x)$ so the equality holds.

### 2.1.5 Linearised Polynomial to Matrix representation

Linearised Polynomials over $F_{q^n}$ can be mapped to their equivalent $n \times n$ matrix over $q$. We are interested here on Linearised Permutation Polynomials, so the equivalent matrix has full rank, it's invertible. In order to convert from the Linearised Polynomial

$$
l(X) = \sum_{i=0}^{n-1} \alpha_i X^{q^i} \quad \alpha_i \in F_{q^n}
$$

Express the Vandermonde matrix with coefficient entries in $F_{q^n}$ as:

$$
M = \begin{pmatrix}
(t^0)^{q^0} & \ldots & \ldots & (t^0)^{q^{n-1}} \\
(t^1)^{q^0} & \ldots & \ldots & (t^1)^{q^{n-1}} \\
\ldots & \ldots & \ldots & \ldots \\
\vdots & \vdots & \vdots & \vdots \\
\ldots & \ldots & \ldots & \ldots \\
(t^{n-1})^{q^0} & \ldots & \ldots & (t^{n-1})^{q^{n-1}}
\end{pmatrix}
$$

And compute $M \cdot \alpha = \beta$ where $\alpha$ is the vector of the coefficients of $l(X)$ such that $\alpha = (\alpha_1, \ldots, \alpha_n) \in F_{q^n}$. In the vector $\beta \in F_{q^n}$ we have the image of the Linearised Polynomial as the vector $\beta = (l(t^0), \ldots, l(t^{n-1}))$. Then take every polynomial evaluation $l(t^i)$ and write it as the $i$ th column in the matrix $L \in F_q^{n \times n}$. Now the matrix $L$ is linear equivalent to the Linearised Polynomial $l(X)$. Moreover, if the matrix $L$ is full rank, it's invertible and its polynomial equivalent $l(X)$ is a Linearised Permutation Polynomial. This process is a well known result as the linear matrix equivalent $L$ acts on the canonical vectors $e_i$ the same way $l(X)$ acts on monomials $t^i$.

#### 2.1.5.1 Optimizing Vandermonde Matrix Computation

The optimization has been done in the code as computing Vandermonde Matrices for Linearised Polynomial interpolation is quite expensive, and if done naively it slows down the implementation. First, Vandermonde Matrix is only used for generating the polynomial $l_2(X)$ which is of degree $q^d$. For that a random $d+1$ vector of coefficients over $F_{q^n}$ is generated. This is, $l_2 = (\alpha_0, \ldots, \alpha_d)$. Then we only need a $n \times (d+1)$ Vandermonde Matrix, only $d+1$ columns instead of $n$. We can view it as a partial Vandermonde matrix that is used to translate linearised polynomials having degree $q^d$ to matrices $n \times n$ over $F_2$. The partial Vandermonde matrix is:

$$M = \begin{pmatrix} (t^0)^{q^0} & \ldots & \ldots & (t^0)^{q^d} \\ (t^1)^{q^0} & \ldots & \ldots & (t^1)^{q^d} \\ \ldots & \ldots & \ldots & \ldots \\ \vdots & \vdots & \vdots & \vdots \\ \ldots & \ldots & \ldots & \ldots \\ (t^{n-1})^{q^0} & \ldots & \ldots & (t^{n-1})^{q^d} \end{pmatrix}$$

- First notice that in every row the powers $(q^0, \ldots, q^d)$ are repeated extensively.
- Then compute every monomial in a precomputed vector such that $(t^{q^0}, \ldots, t^{q^d})$
- For every $i$ th row and $j$ th column, compute $(t^{q^j})^i$ by taking the $j$ th element of the precomputed vector, so we don't have to raise to $q^j$.
- When the $i$ th row is divisible by a small prime factor $r$, divide $\frac{i}{r} = s$ and exponent every monomial of the $s$ th row to the power of $r$ to get the $i$ th row easily. Small prime factors are $2, 3, 5$.

These steps guarantee an efficient computation of the Vandermonde matrix.

#### 2.1.5.2 Selecting linearised polynomials with maximal Degree

The scheme uses Linear Algebra and Tensor representation, so invertible matrix $L_1$ is randomly generated over $F_2$. It's not randomly generated as a polynomial in $F_{q^n}[X]$ so at first glance we don't know what degree does $l_1(X)$ have as a linearised polynomial. We only know that the matrix $L_1$ is the matrix representation of the polynomial $l_1(X)$. HPPC works with an HFE polynomial with maximal degree bounded by $D = q^{n-1}$ which is only achieved if the selected random matrix $L_1 \in F_2^{n \times n}$ has a representation $l_1(X)$ with degree $D = q^{n-1}$. Checking the degree would require using the Vandermonde matrix inverse to obtain the coefficient tuple of $l_1(X)$ and check if there's an element in the $n$-th position, so

degree $q^{n-1}$. This is overkill, we want to cut time from here so we stick to linear algebra in $F_2$ instead of computing in the polynomial extension. The approach here is **to prove** that selecting a random invertible matrix $L_1$ over $F_2$ guarantees that its linearised polynomial equivalent $l_1(X)$ has degree $q^{n-1}$, this is having monomial $\alpha_{n-1}X^{q^{n-1}}$.
This is done by finding the number of polynomials $l_1(X)$ having degree $q^{n-1}$. The number of polynomials $\sum_{i=0}^{n-1} \alpha_i X^{q^i} = \alpha_0 X^{q^0} + \ldots + \alpha_{n-1}X^{q^{n-1}}$ that have degree $q^{n-1}$ is given by:

$$(\prod_{i=1}^{n-1} q^n) \cdot (q^n - 1) = (q^{n^2-n}) \cdot (q^n - 1) = q^{n^2} - q^{n^2-n} = q^{n^2-n}(q^n - 1)$$

The aforementioned expression counts how many *linpolys* do exist with degree $q^{n-1}$, to compute the probability of selecting such polynomials at random:

$$\frac{q^{n^2-n}(q^n-1)}{q^{n^2}} = \frac{q^n-1}{q^n}$$

When setting $q = 2$ the limit of the ratio tends to one, so there exists a high probability of selecting a random $n \times n$ invertible matrix $L_1$ over $F_2$ such that its linearised polynomial representation has maximal degree $q^{n-1}$. With this in mind, there is no need to check for degree and invertible $L_1 \in F_2^{n\times n}$ can be selected at random. Concluding that partial Vandermonde matrix $n \times (d+1)$ is used only for generating $l_2(X), L_2$ and there is no need to translate matrix $L_1$ to a linearised polynomial $l_1(X) \in F_{q^n}[X]$ to check for maximal degree. This results serves for lowering key generation time as reflected in the implementation code.

## 2.2 HPPC Scheme

In MPKC we are interested on functions $P(X) = T \circ F \circ S(X)$ such that $T, S$ are linear/affine maps and $F(X)$ is a quadratic set of equations. This is because $P(X)$ is non-linear thanks to the internal structure of $F(X)$. However, we need $F(X)$ to be a trapdoor function since recovering $X$ from the public key $P^{-1}(Y) = X$ is considered hard.

To build such trapdoor functions a new family of private polynomials $F(X)$ is presented. The construction guarantees that is easy to evaluate the map but hard to recover the original point, in theory. Let's give a detailed description

### 2.2.1 Description

All the operations are done in the base field $q = 2$, where the extension field has $q^n$ elements. In order to construct a theoretical secure function $P(X) = T \circ F \circ S(X)$ we select linear or affine transformations $T, S \in F_2^{n\times n}$. Then select Linearised Permutation Polynomials $l_1(x), l_2(x) \in F_{q^n}[X]$ where $L_1, L_2 \in GL(2, n)$ is the matrix representation of $l_1(x), l_2(x)$. $l_2(X)$ must be monic, where the highest monomial is $X^{q^d}$. The degree of

Linearised Permutation Polynomials $l_1(x)$ is at most $q^{n-1}$ and for $l_2(x)$ is $q^d$, for $d = 10$. Then

$$\begin{aligned} M &= ({C_{f(x)}}^0, \ldots, {C_{f(x)}}^{n-1}) \\ F &= L_1 \otimes (L_2 \cdot L_1) \\ P(\overrightarrow{x}) &= T \cdot M \cdot \underbrace{(L_1 \otimes (L_2 \cdot L_1))}_{F} \cdot (S \otimes S) \cdot (\overrightarrow{x} \otimes \overrightarrow{x}) \end{aligned}$$

And $P(\overrightarrow{x})$ is the $n \times n^2$ matrix representation of the public key polynomial.

To compute the trapdoor function $G(X)$ we do the composition

$$\begin{aligned} &G(X') = P(S^{-1} \circ L_1^{-1}(X)) \\ &= T \cdot M \cdot (L_1 \otimes (L_2 \cdot L_1)) \cdot (S \otimes S) \cdot (S^{-1} \otimes S^{-1}) \cdot (L_1^{-1} \otimes L_1^{-1}) = \\ &= T \cdot M \cdot (L_1 \otimes (L_2 \cdot L_1)) \cdot (L_1^{-1} \otimes L_1^{-1}) = \\ &= T \cdot M \cdot (I_n \otimes (L_2)) \end{aligned}$$

This is equal to the monic polynomial $G(X) = X * l_2(X)$ having degree $q^d + 1 = D = 1025$

Then $G(X') = P(X) = Y$ and if we solve the system $G(X') - Y = 0$ over $F_{q^n}[X]$ we recover $P(X) = Y$ as $S^{-1} \cdot L_1^{-1} X' = X$.

### 2.2.2 Key Generation

The scheme relies its security on the construction of a private key polynomial $F(X) = l_1(X) * (l_2 \circ l_1(X))$ which is a product of two Linearised Permutation Polynomials. The polynomial $l_1$ has at most degree $q^{n-1}$ and $l_2$ has degree $q^d$, then its product $F(X) \in F_{q^n}[X]$ has big degree, which is desirable to resist rank attacks at a first glance, as the resulting polynomial doesn't' have low degree compared to other BigField schemes like: HFEv-, QUARTZ, and GeMSS [Cas].

#### 2.2.2.1 Public Key Construction

In order to generate a public key we must set-up the parameters of the scheme:

- Let $q = 2$, $f(x) \in F_q[x]$ an irreducible polynomial over $q$, then the Finite Field $F_{q^n}$ has $q^n$ elements.
- Select invertible linear matrices $T, S \in F_q^{n \times n}$ which are used to hide the private polynomial map $F(X)$.

- Select $L_1 \in F_q^{n \times n}$ as an invertible matrix over $q$. It's representation in $F_{q^n}[X]$ guarantees $l_1(X)$ to be a Linearised Permutation Polynomial with highest degree monomial at most $\alpha X^{q^{n-1}}$

- Select $l_2(X) \in F_{q^n}[X]$ as a **monic** Linearised Permutation Polynomial with degree $q^d$, for $d = 10$ so degree $2^{10} = 1024$ .

- For that generate a random (monic) vector $\alpha = (\alpha_0, \dots, \alpha_d)$ with $\alpha_d = 1$, having $d + 1$ coefficients.

- Now $\alpha$ is the coefficient representation of the monic Linearised Polynomial $l_2(X) \in F_{q^n}[X]$ with degree $q^d$.

- Compute the partial Vandermonde matrix $M$ of size $n \times (d + 1)$ and multiply it by $\alpha$ to obtain $M \cdot \alpha = v \in F_{q^n}$.

- Represent the output vector $v$ as the matrix $L_2 \in F_q^{n \times n}$.

- Compute the rank of $L_2$ over $q$. If it's full rank then $l_2(X)$ is a Linearised Permutation Polynomial and $L_2 \in F_q^{n \times n}$ is its matrix representation.

- Compute the matrix representation of multiplication of elements in the Finite Field $F_{q^n}$ as $M = ({C_{f(x)}}^0, \dots, {C_{f(x)}}^{n-1})$, where ${C_{f(x)}}^i$ is the $i$th power of the Companion Matrix of the irreducible polynomial $f(x)$.

- Compute the multilinear private map $F = L_1 \otimes (L_2 \cdot L_1)$ which is the tensor representation over $q$.

- Compute the trapdoor polynomial map $G(X) = X * l_2(X) \in F_{q^n}[X]$ which is of degree $q^d + 1 = 1025$ and is defined over $F_{q^n}$. It's mandatory to compute it over the finite field extension as Berlekamp's Algorithm cannot be applied to the Tensor Representation. Here the author is unaware of a root solving algorithm reduction to Tensor Algebra.

- Compute the Tensor representation of the Public Key over $q$ as $P(X) = T \cdot M \cdot F \cdot (S \otimes S) \cdot (x \otimes x)$.

#### 2.2.2.2 Private Key Construction

The owner must retain matrices $T^{-1}, S^{-1}, L_1^{-1}$ and the private polynomial $G(X) = X * l_2(X)$.

### 2.2.3 Signing process

#### 2.2.3.1 Message Signing

- For signing a message $m$ of any size compute the digest of the message via a Hash Function $H(m) = y$. Truncate the output of the digest to fit in a $n$ bit vector, if necessary.
- Here $v \in F_q^n$ is a vinegar vector with the first $k = 8$ positions randomized and the rest $n - k$ positions set to zero.
- To solve the equation $P(x) + v = y$ start by selecting random $k$ values for the vinegar vector $v$.
- Compute $y' = y - v$ and apply $T^{-1} \cdot y' = z$.
- Express the vector $z$ as it's polynomial representation as $Z = \phi^{-1}(z) \in F_{q^n}$.
- Recover $X' \in F_{q^n}$ by root finding (Berlekamp's) on the polynomial $G(X') - Z = 0$.
- If no root is found for $G(X') - Z = 0$ then go back to step 2.
- Once the polynomial $X'$ is recovered, express it as the vector $x' = \phi(X') \in F_q^n$.
- Compute the vector $S^{-1} \cdot L_1^{-1} \cdot x' = x$ which is the signature point.

#### 2.2.3.2 Message Verification

- The verifier must posses the Public Key in Tensor form $P(x) = A \cdot (x \otimes x) = y$.
- Verifier receives a triplet $(H, x, v, m)$ where $H$ is the Hash function, $x$ is the signature, $v$ is the vinegar vector and $m$ the message to be validated.
- Verifier takes vinegar vector $v \in F_2^k$ and forms the vector $v' = (v_1, \ldots, v_k, 0, \ldots, 0) \in F_2^n$.
- Verifier computes $P(x) + v' = H(m) = y$ and if its correct the signature is trusted as only the owner of the private key can issue valid signatures.

### 2.2.4 Key Sizes

The advantage of MPKC over other PQC candidates is the reduced signature bit length. However, it's been widely commented that MPKC has notorious trade-off between the signature bit length and public/private key pairs (specially HFE variants). Let's examine these cases:

#### 2.2.4.1 Public Key

Recall that the Tensor representation of the Public Key is $P(x) = T{\cdot}M{\cdot}F{\cdot}(S{\otimes}S){\cdot}(x{\otimes}x) = A{\cdot}(x{\otimes}x)$. The matrix $A \in F_q^{n\times n^2}$ is the public key that everyone sees and has size $log_2 q{\times}n^3$. However, it's been optimized to $log_2 q \times n \times \frac{n(n+1)}{2}$

#### 2.2.4.2 Optimized Public Key size

When dealing with tensor products of vectors $x \otimes x$ we encounter monomials $x_i x_j$ and $x_j x_i$, meaning that the tensor product of two $n$ bit length vectors is of size $n^2$. The size of this vector can be reduced to $\frac{n*(n+1)}{2}$ by skipping the redundant terms. The same for the public key matrix, which is of dimension $n \times n^2$ is reduced to $n \times \frac{n(n+1)}{2}$. To compress the public key note that each row is the vectorisation of a quadratic form $Q_i$. Quadratic Forms are symmetric in the sense that coefficients can be packed since any resulting quadratic monomial can be expressed as $a_{i,j}x_i x_j + a_{j,i}x_j x_i$. This facts leads to the compression of the public key matrix to a rectangular matrix of size $n \times \frac{n(n+1)}{2}$.

#### 2.2.4.3 Private Key

The private key is comprised of matrices $T^{-1}, S^{-1}, L_1^{-1} \in F_q^{n\times n}$ and the polynomial $G(X) = X * l_2(X) \in F_{q^n}[X]$ of degree $q^d + 1$. Then the Private Key size sums up to $log_2 q \times (3n^2 + (d+1) \times n)$.

#### 2.2.4.4 Signature Length

The input vectors $(x, v)$ have length $n$ and $k$ respectively. Then the signature size is $\log_2 q \times (n + k)$. When $n = 128$ a random salt of 32 bits must be generated an appended to the signature. Check parameter list for more info.

### 2.2.5 Key Encoding and Decoding

NIST proposed a template for KAT values where Public and Private keys are represented as an unsigned char vector, this is, a byte vector. As we are dealing with matrices over $F_2$, literally 1's and 0's we must pack every 8 bit into a byte thus encoding the bit string into a byte vector. This is done in the code by dividing every row of a matrix in 8 bit groups, then packing every group into a decimal value from $0 - 255$.

The reverse operation is decoding. We must convert from unsigned char or a byte vector to a matrix. For that every byte from the vector is represented as a 8 bit string, its binary representation. That representation is copied directly to the rows of the matrix.

Both methods guarantee that Public and Private Keys along with signatures are expressed in Hexadecimal representation.

# 3 List of parameter sets

First define every parameter of the tuple $(q, n, k, D)$.

- $q$ is the base field of the Finite Field.
- $n$ is the dimension of the Finite Field and the Vector space which is constructed by using Tensor Algebra.
- $k$ is the number of vinegar variables. Recall that $v \in F_q^n$ is a $n$ vector having $k$ entries in $F_q$ and other $n - k$ entries set to zero.
- $d$ is the exponent used for constructing the degree $D$.
- $D$ is the degree of the polynomial $G(X) = X * l_2(X)$ which is monic, this is, largest monomial is $X^{q^d+1}$.

| Scheme | Security Level | q | n | k | D | \| pk \| (KB) | \| sk \| (KB) | sign (B) |
|---|---|---|---|---|---|---|---|---|
| HPPC128 | 2 | 2 | 128 | 8 | 1025 | 129 | 6.17 | 21 |
| HPPC192 | 4 | 2 | 192 | 8 | 1025 | 434.25 | 13.75 | 25 |
| HPPC256 | 5 | 2 | 256 | 8 | 1025 | 1028 | 24.34 | 33 |

Table 1: Parameter list for Security Levels KB:Kilo-Byte, B:Byte

# 4 Detailed performance analysis

## 4.1 Testing Platform

The reference and optimized implementation has been tested on a single platform.

| Computer | Processor | Frequency (MHz) | Max freq. (MHz) |
|---|---|---|---|
| Workstation | AMD Ryzen 3700x | 3600 | 4200 |
| Embedded | Raspberry PI 4b | 1500 | 1500 |

Table 2: Description of the testing platform.

| Computer | OS | Kernel | RAM |
|---|---|---|---|
| Workstation | Arch Linux | 6.1.12 | 32GB |
| Embedded | OpenBSD 7.3 | GENERIC.MP | 4GB |

Table 3: Description of OS and RAM

## 4.2 Third Party Open Source Libraries

The reference and optimized implementation make use of three libraries: FLINT, M4RI and NTL. Both FLINT and M4RI are supported in ANSI C, however NTL is only usable in C++. So the code has been separated, exporting NTL functionality to a single .cpp and .hpp file, which is minimal and can be replaced in the future to complete the whole in ANSI C.

- FLINT is used for the selection of the matrix $L_2$ and the private trapdoor function $G(X) = X * l_2(X)$. This is done by the Vandermonde matrix representation, thus expressing matrix $L_2$ of rank $n$ as the representation of the polynomial $l_2(x)$, which is a Linearised Permutation Polynomial.
- M4RI is used for dealing with operations of linear algebra over $F_2$ (mult, sum, rank, vectors, matrices) thus for constructing Private and Public Key and for obtaining-verifying signatures using Tensor Algebra.
- NTL is used only for finding roots of the polynomial $G(X)-Y = 0$ as FLINT struggles either when finding the roots of the polynomial or when splitting a single factor. In this case NTL outperforms FLINT. For more information refer to [Sho].

### 4.2.1 Differences between Operating Systems

There are major differences between the packaging found among Unix-like OS and GNU/Linux distributions.

#### 4.2.1.1 GNU/LINUX

In Arch Linux and Kali (Debian based) FLINT is at version 2.9.0 which guarantees that some primitives like *fmpz_mod_mat_rank(), fmpz_mod_mat_inv* do exist. The implementations do rely on these primitives which are not present for example in Ubuntu (Desktop) 22.04 and Debian 11. Manual installation of FLINT 2.9.0 is mandatory in those cases since compilation with GCC of implementations will throw errors as it cannot find those function calls. In the case of M4ri and NTL both libraries are integrated in apt and pacman (package managers).

#### 4.2.1.2 BSD

In the case of other Unix-Like OS, OpenBSD has been tested on a Raspberry PI 4b (aarch64). FLINT was manually compiled and installed with gmake ang gcc. The implementations compiled by passing the argument ”-I /usr/local/includes” so GCC locates FLINT’s header files. NTL has a *ports* section so it can be intregrated easily and M4ri must be manualy compiled and installed. As libstdc++ is required and not found in this system, Clang++ is used to compile the code using libc++, the C++ runtime library supported by the OpenBSD team.

## 4.3 Reference vs Optimized implementation

The reference and optimized implementations are the same at code level. The reference implementation runs in single core. However the MAKEFILE includes an optimized version that selects Strassen Matrix multiplications using M4RI’s method mzd_mul() that applies the method plus it can parallelize operations. This parameter is disabled by default and it’s experimental. It can improve KeyGen method as Tensor Products are better computed in parallel than in a single core.

## 4.4 Benchmark

The MAKEFILE has two tests. One is the Fast test in single core, the other the Strassen fast test which may exploit parallelization. KAT procedure is not contemplated here, as it's compiled without optimizations, just for collecting and satisfying NIST tests.

### 4.4.1 Fast test

Here the Fast test in single core is measured on *Workstation* and *Embedded* computers.

| Scheme | Gen | Sign | Verify |
|---|---|---|---|
| HPPC128 | 539.26ms - 542.67ms | 689.75ms - 725.33ms | 3.18ms - 3.2ms |
| HPPC192 | 3.03s - 3.05s | 1s - 1.7.s | 10.7ms - 10.9ms |
| HPPC256 | 9s - 9s | 1.2s -2.4s | 25.1ms - 25.2ms |

Table 4: Median - Average time of distinct Parametrizations in 20 rounds in *Workstation*

| Scheme | Gen | Sign | Verify |
|---|---|---|---|
| HPPC128 | 2.3s - 2.5s | 2.57s - 2.85s | 9.12ms - 9.13ms |
| HPPC192 | 16s - 16.3s | 2.4s - 5.4s | 29.84ms - 33ms |
| HPPC256 | 49.2s - 50.4s | 10.5s -11.7s | 80.2ms - 83.33ms |

Table 5: Median - Average time of distinct Parametrizations in 20 rounds in *Embedded*

### 4.4.2 Strassen Fast test

Here the Strassen fast test is measured. Notice that it may run in parallel.

| Scheme | Gen | Sign | Verify |
|---|---|---|---|
| HPPC128 | 405.1ms - 407.3ms | 700ms - 720.1ms | 3.19ms - 3.2ms |
| HPPC192 | 2s - 2.1s | 1.2s - 1.7ms | 11.47ms - 11.5s |
| HPPC256 | 6s - 6.2s | 1.2s - 2.5s | 24ms - 24.9ms |

Table 6: Median - Average time of distinct Parametrizations in 20 rounds in *Workstation*

| Scheme | Gen | Sign | Verify |
|---|---|---|---|
| HPPC128 | 1.3s - 1.5s | 1.93s - 2.95s | 8.50ms - 9.03ms |
| HPPC192 | 13.4s - 13.6s | 2.5s - 4.89s | 29.4ms - 29.97ms |
| HPPC256 | 45.8s - 46.1s | 8.28s - 11.9s | 80.1ms - 81.32ms |

Table 7: Median - Average time of distinct Parametrizations in 20 rounds in *Embedded*

## 4.5 Workstation vs Embedded

The results of the execution of both tests in the workstation and embedded lead to the conclusion of the workstation having more than 3 times speedup in computation time per operation mode (Sign, Verify and Gen). Still the embedded system performs the operations in time, at least for HPPC128.

The other interesting result is that the Strassen option has noticeable speedup on both machines, which is passed as an optional compiler flag to turn on Strassen's method for matrix multiplication, reducing computation time in matrix tensor operations.

In resume, **workstations** comply for generating keys and message signing-verifying in time and **embedded** systems are capable of performing all the operations, however, the one that escalates well in time is **verifying**.

# 5 Expected Strength

## 5.1 EUF-CMA security

The presented scheme must be analyzed from the perspective of the **EUF-CMA** security model applied to digital signatures. The model proposes the following conditions:

- Challenger $\mathcal{C}$ generates a pair of public-private keys $(pk, sk)$ and sends the public key $pk$ to the adversary $\mathcal{A}$.
- The adversary $\mathcal{A}$ has access to the oracle and queries for the message $m$.
- The oracle returns the signature $\theta \leftarrow \text{Sign}(sk, m)$ and stores the message $m$ into the message list $\mathcal{Q}$, so every submitted message by $\mathcal{A}$ has to be not repeated or it will be discarded by the Oracle and/or the Challenger $\mathcal{C}$.
- $\mathcal{A}$ wins when finds a valid pair $(m^*, \theta^*)$ where $\text{Verify}_{pk}(m^*, \theta^*) = 1$ and $m^* \notin \mathcal{Q}$, this is, the message $m^*$ must not be submitted to the oracle and $\theta^*$ is a valid signature for $m^*$.

Given the message $m^* \notin \mathcal{Q}$ the adversary $A$ looks for solving the equation $P(\theta^*) - v = H(m^*)$, where $v \in F_2^n$ has first $k$ randomized entries and last $n - k$ entries set to zero. Obtaining such signature is as hard as solving the underlying MQ problem.

### 5.1.1 Security

For example if we don't restrict the vector $v \in F_2^n$ to have $n - k$ entries set to zero, the adversary $\mathcal{A}$ computes $H(m^*)$ and selects a random $\theta^* \in F_2^n$ such that $v = P(\theta^*) + H(m^*)$. Then the signature pair $(\theta^*, m^*, v, H)$ is valid as $\text{Verify}_{pk}(m^*, \theta^*) = P(\theta^*) + v = H(m^*)$. However, for $k = 8$ if we restrict the vector $v$ to have last $n - 8$ entries set to zero and the first 8 entries randomized the thing changes for the adversary $\mathcal{A}$. Like in the previous case, the adversary computes $H(m^*)$ and selects a random $\theta^* \in F_2^n$ such that $v = P(\theta^*) + H(m^*)$. But notice that now $v$ doesn't have the mentioned structure, this is, the last $n - 8$ entries are not set to zero so the verifier rejects the signature.

### 5.1.2 Signature forgery

The only way for $v$ having the last $n - 8$ entries set to zero is to have a collision on the last $n - 8$ entries of $P(\theta^*)$ and $H(m^*)$ and to find such collision the attacker must solve a system of $n - 8$ quadratic equations on $n$ variables. The attacker must solve $\overline{P(\theta^*)} + \overline{H(m^*)} = 0$ where the over-line in the equations indicates the selection of the last $n - 8$ equations of $P(\theta^*)$ and the last $n - 8$ entries of the hash $H(m^*)$. Solving such system equals to solving the underlying MQ problem at a minimum of 120 equations in 128 variables. Selecting random values for $(x_{120}, \ldots, x_{128}) \in F_2^8$ yields a $m = n = 120$ equation system which can be translated to a semi-regular sequence adding equations $x_i^2 + x_i = 0$ for applying $F_4$ algorithm. In general the complexity of this system shows that $d_{reg} \leq 16$ [Sal] which is a good indicative that the following analysis is consistent. Head to [SH] for a formal proof on schemes HFE and UOV.

# 6 Analysis of known attacks

As the scheme belongs to BigField family, numerous attacks can be ruled out on HFE. Here the goal is to estimate the complexity of these attacks applied to the parametrization of the presented scheme.

## 6.1 Index & Degree Regularity

There are distinct cases for estimating attacks using Gröbner in multivariate polynomial equation systems. For example, under some conditions, over-defined systems are easier to solve than under-determined or systems where $m = n$ which is the case of **HPPC**. The goal is to demonstrate that a system belongs to the *worst-case family* of polynomials such that the computed Gröbner basis has (almost) maximal degree of regularity, this is $d_{reg} \leq \#MB = \sum_{i=1}^{n}(d_i - 1) + 1$, where MB is the *Macaulay Bound*, an upper bound that defines the highest degree that a term can have in the resulting Gröbner basis.

In general, for regular systems where $m = n$ the index of regularity plus one coincides with the upper bound MB so we conclude that $d_{reg} \leq i_{reg} + 1$. The *index of regularity*, $i_{reg}$ is the degree of $HS_{\mathcal{I}}(t) = \frac{\prod_{i=1}^{n}(1-t^{degp_i})}{(1-t)^n}$, the Hilbert Series polynomial of the Ideal generated by the polynomials of the system $P(X) - Y = 0$.

### 6.1.1 HFEv schemes

The Degree of Regularity calculation of HFE, HFE- and HFEv- polynomials was previously studied in [Fau] [FJ] [DK] [Sal]. Also in [DK] authors mention that using a binary extension field has no effect as HFE can be carried out to $q = 2$ lowering the degree of regularity at the cost of having a system with more variables and equations. In [DK] set $d = \lfloor \log_2(D) \rfloor$ so the upper bound for the degree of regularity for an HFEv polynomial is estimated as:

$$d_{reg} \leq \frac{(q-1)(d+v-1)}{2} + 2$$

Recently NIST candidate GeMSS [Cas] (QUARTZ based HFEv- variant) was considered broken where parametrization must be tuned in order to achieve security [PD] [STV]. Hence the ”minus” modifier and vinegar variables do not enhance security, just by a polynomial factor. In the presented scheme no ”minus” modified is used (nude HFE). A minimum vector of vinegar variables is used to guarantee finding a root of the trapdoor polynomial $G(X)$ through distinct trapdoor inversions.

### 6.1.2 Semi-regular sequences

Solving multivariate equations over $q = 2$ has the advantage of adding $n$ equations of type $x_i^2 + x_i = 0$, resulting in a quadratic system of $2n$ equations in $n$ variables, which is a *semi-regular* sequence of multivariate polynomials.
It results that there are estimations in Bardet's work that can be applied for quadratic systems over $F_2$ where $m = n$ [Sal], the case of **HPPC**. This estimation is called $d_{max}$. In [DY] authors state that Bardet's estimations won't work when some structure is introduced to the generated system. For example an HFE polynomial with $n = 80$ has $d_{max} = 12$ however if the degree of the polynomial is $D = 2^9 + 1 = 513$ then $\frac{(q-1)(d+v-1)}{2} + 2 \approx 6$ which is the half of $d_{max}$.

### 6.1.3 Degree of regularity for HPPC

Let $m = 2n$ as the result of adding $n$ equations $x^2 + x_i = 0$ to the system. The Hilbert series for quadratic *semi-regular* sequences of $2n$ equations in $n$ variables is given as [Sal]:

$$HS_{\mathcal{I}}(t) = \frac{\prod_{i=1}^{2n}(1 - t^2)}{(1 - t)^n}$$

The value for $d_{max}$ equals the degree of the first non-positive coefficient and serves for a conservative bound.

- For $n = 128$ first non-positive term is $-6962621258288688000t^{17}$
- For $n = 192$ first non-positive term is $-759963592123628272114009536t^{23}$
- For $n = 256$ first non-positive term is $-92341817249200423510805160209529600t^{29}$

The degree of the central HFE polynomial $F(X)$ is at most $q^{n-1}$ which is big enough for resisting against well-known attacks. The trapdoor polynomial $F(S^{-1} \circ L^{-1}(X)) = G(X') = X' * l_2(X')$ has degree $q^d + 1$ instead, enough for Berlekamp's algorithm to work for root finding. For $n = 128$ the HFE polynomial has at most degree $D = 2^{127}$ which gives an upper bound of $d_{reg} \leq 63$ using concepts from [DY].

## 6.2 Gröbner Basis and $F_4$ algorithm

The MQ problem in MPKC is based on finding the set of roots over $F_2$ from the Tensor representation of the public key $P(x) - y = 0$. The normal approach is to solve it by using Gröbner Basis, which finds an Algebraic Variety that contains the roots (solution) of the system. This process is bounded by the nº of variables and the degree of regularity.
Testing has been conducted on instances of HPPC where $11 \leq n \leq 32$ using Wolfram Mathematica for generating the symbolic public key and SAGE for computing Gröbner basis and $d_{reg}$. Equations $x_i^2 + x_i = 0$ are appended to the polynomial $P(X) - Y = 0$ and stored in a text file, then loaded via Sage to generate a Gröbner basis. Here $d_{reg}$ is the *degree of semi-regularity* which is bounded by $d_{max}$ [Sal]. It results that SAGE reports the same *degree of semi-regularity* as $d_{max}$ for each parameter $n$.
The output basis is always linear in variables $(x_1, \ldots, x_n)$ for $n = 2^k$. For other values of $n$ quadratic monomials $x_i * x_j$ have been observed in the output basis. It's an open question if the selected parametrization is the most optimal from the point of view of security.
The following table represents the degree of regularity for each instance of HPPC for the parameter list previously given.

| Scheme | Ding-Yang [DY] | Bardet [Sal] |
|---|---|---|
| HPPC128 | 70 | 17 |
| HPPC192 | 102 | 23 |
| HPPC256 | 134 | 29 |

Table 8: Analysis of $d_{reg}$ in distinct studies

The degree of regularity serves for estimating the required number of field operations for computing a Gröbner basis, which is:

$$\mathcal{O}(\binom{n + d_{reg}}{d_{reg}}^w)$$

| Scheme | Ding-Yang [DY] | Bardet [Sal] |
|---|---|---|
| HPPC128 | $2^{360}$ | $2^{145}$ |
| HPPC192 | $2^{536}$ | $2^{204}$ |
| HPPC256 | $2^{712}$ | $2^{264}$ |

Table 9: Complexity for distinct $d_{reg}$ setting $w = 2$

## 6.3 MinRank

The MinRank problem MR$(m, n, r)$ consists of finding a linear combination of $m$ matrices of size $n \times n$ to obtain a $n \times n$ matrix of rank $r$. MinRank problem is NP-complete [Bus].

### 6.3.1 Attacks

- The first one was the Kipnis-Shamir attack [KS], where the HFE public key is computed as Quadratic Form $x^T Q x \in F_{q^n}$. From here linear/affine transformations $T$ and $S$ are computed by polynomial solving applying the Relinearization technique.

- The linear algebra technique, which was presented by Goubin and Courtois [GC] was applied to the triangular system TTM. It finds a linear combination $M = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_j M_j X_i = 0$, where matrices $M_j$ and vectors $X_i$ have been previously defined.

- The Minor's attack [PD] is more popular in literature and is applicable to other families [STV] where elements of the underlying algebraic structure are not defined in a Finite Field (base or extension). The goal is to obtain a Matrix with rank $r$ where the determinant of submatrices of size $(r+1) \times (r+1)$ vanish, these are the minors of rank $r+1$. This is a well-known property on Linear Algebra, as a matrix $M$ of rank $r$ has rank $r$ only if it's $r+1$ minors do vanish. The estimation from [PD] is used for complexity calculation:

$$\mathcal{O}(\binom{n+d+v+1}{d+1}^{w})$$

Minor's method for HFE depends entirely on the rank of the central polynomial $F(X)$ which is at most $d = \log_2(D) = n - 1$.

| Scheme | $(n, d, v)$ | Minors |
|---|---|---|
| HPPC128 | $(128, 127, 8)$ | $2^{518}$ |
| HPPC192 | $(192, 191, 8)$ | $2^{774}$ |
| HPPC256 | $(256, 255, 8)$ | $2^{1030}$ |

Table 10: Complexity for Minor's method

### 6.4 Conclusion

$F_4$ algorithm is exponential in $d_{reg}$ where Minors seems intractable as the degree $D$ of central HFE polynomial grows. In literature covered cases for HFE variants are those with $D$ is small. At the moment HPPC is expected to be resistant to existing attacks for the selected parametrization.

## 7 Advantages and limitations

### 7.1 Advantages

- **Small Signatures:** Schemes based on Multivariate Cryptography are well known for their small signature size. Signatures are sent along with the message to the verifier, so it doesn't take much bandwith over a network.
- **Fast Verification:** The verification of a signature is really fast for all the covered parameters.
- **Simplicity:** The mathematics behind the scheme are based on concepts found in BigField schemes like HFE, which is a very well known scheme, counting with multiple variations and cryptanalytic techniques.
- **Arithmetic:** The operations done by the scheme are easily handled by any electronic device as the scheme mainly relies in Linear Algebra over $F_2$ and operations in a binary Finite Field.

### 7.2 Limitations

- **Public key size:** The size of the compressed Public Key is $n \times \frac{n(n+1)}{2}$ thus bigger than other PQC schemes.
- **Key Generation:** Tensor product of matrices over $F_2$ is a costly operation when their dimension is relatively big, which is the case. Optimization must be done to cut time from the key generation.
- **Signing time:** The signature process must find the roots of the polynomial equation $G(X) - Y$ over a Finite Field. This process may not yield a root, so new vinegar values must set-up and the equation must be solved again. This is a limitation of the private polynomial of HFEv- schemes like QUARTZ and GeMSS. This is because the private polynomial is not bijective, thus needing to change the constant term using new vinegar values, in order to find a new root.

## References

[Patb] Jacques Patarin. *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms.* URL: http://www.minrank.org/hfe.pdf.

[Cas] J.C. , Macario Rat , G. Patarin J. , Perret-L. , Ryckegem-J Casanova A. , Faugere. *GeMSS: a great multivariate short signature.* URL: https://www-polsys.lip6.fr/Links/NIST/GeMSS_specification.pdf.

[WP] Christopher Wolf and Bart Preneel. *Taxonomy of Public Key Schemes based on the problem of Multivariate Quadratic equations.* URL: https://eprint.iacr.org/2005/077.pdf.

[TH] T.Matsumoto and H.Imai. *Public Quadratic Polynomial-tuples for eficient sigriature-verification and message encryption.* URL: https://link.springer.com/chapter/10.1007/3-540-45961-8_39.

[Pata] Jacques Patarin. *Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'98.* URL: https://link.springer.com/article/10.1023/A:1008341625464.

[KS] Aviad Kipnis and Adi Shamir. *Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization.* URL: https://link.springer.com/chapter/10.1007/3-540-48405-1_2.

[STV] John Baena , Pierre Briaud , Daniel Cabarcas , Ray Perlner , Daniel Smith-Tone and Javier Verbel. *Improving Support-Minors Rank Attacks: Applications to GeMSS and Rainbow.* URL: https://eprint.iacr.org/2021/1677.pdf.

[TV] Magali Bardet , Maxime Bros , Daniel Cabarcas , Philippe Gaborit , Ray Perlner , Daniel Smith-Tone , Jean-Pierre Tillich and Javier Verbel. *Improvements of Algebraic Attacks for solving the Rank Decoding and MinRank problems.* URL: https://arxiv.org/pdf/2002.08322.pdf.

[Bus] J.O. Shallit , G.S. Frandsen , J.F. Buss. *The computational complexity of some problems of linear algebra.* URL: https://www.brics.dk/RS/96/33/BRICS-RS-96-33.pdf.

[Sho] Victor Shoup. *NTL vs FLINT.* URL: https://libntl.org/benchmarks.pdf.

[Sal] Magali Bardet , Jean-Charles Faugère , Bruno Salvy. *Complexity of Gröbner basis computation for Semi-regular Overdetermined sequences over F2 with solutions in F2.*

[SH] Koichi Sakumoto , Taizo Shirai and Harunaga Hiwatari. *On Provable Security of UOV and HFE Signature Schemes against Chosen-Message Attack*. URL: https://link.springer.com/chapter/10.1007/978-3-642-25405-5_5.

[Fau] J.-C. Faugère. *A new efficient algorithm for computing Gröbner bases (F4)*. URL: https://www.sciencedirect.com/science/article/pii/S0022404999000055.

[FJ] Jean-Charles Faugère and Antoine Joux. *Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Gröbner bases*. URL: https://link.springer.com/chapter/10.1007/978-3-540-45146-4_3.

[DK] Jintai Ding and Thorsten Kleinjung. *Degree of Regularity for HFE*. URL: https://eprint.iacr.org/2011/570.pdf.

[PD] Chengdong Tao , Albrecht Petzoldt and Jintai Ding. *Improved Key Recovery of the HFEv- Signature Scheme*. URL: https://eprint.iacr.org/2020/1424.pdf.

[DY] Jintai Ding and Bo-Yin Yang. *Degree of Regularity for HFEv and HFEv-*. URL: https://link.springer.com/chapter/10.1007/978-3-642-38616-9_4.

[GC] Louis Goubin and Nicolas Courtois. *Cryptanalysis of the TTM Cryptosystem*. URL: http://www.goubin.fr/papers/ttm.pdf.