Multivariate Cryptographic Primitive based on the product of the roots of a polynomial over a field

Borja Gómez
kub0x@elhacker.net

October 15, 2020

Abstract

Cryptographic Primitives in Multivariate Public Key Cryptography are of relevant interest, specially in the quadratic case. These primitives classify the families of schemes that we encounter in this field. In this paper, the reader can find a new primitive based on the product of the roots of a polynomial over a field, where the coefficients of this polynomials are the elementary symmetric polynomials on n variables, which guarantees a solution when inverting the scheme. Moreover, a cryptosystem and a digital signature scheme are built on top of this primitive, where distinct parametrizations and criteria that define the schemes are commented, along with applications of attacks available in literature.

1 Introduction

A small summary is done here. The cryptographic primitive is introduced in plus the cryptosystem and digital signature are explained in section 2, section 3 and section 4. There are several cases of parametrizations that are only commented but not developed during this research. Continuing, the primitive supports a special parametrization that allows the cryptographer to obtain a system of equations having degree d over F_p for $d = 2$ in section 5, which guards similarity with (un)balanced Oil-Vinegar schemes, but it's internals are completely different. The public key's size complexity is commented in section 6.

Multiples strategies for speeding up the Public Key generation are developed in section 9, which results in ≈ 40 seconds to compute the public key for the recommended parametrization . In section 7 the relevancy of basing the scheme in the IP2 assumption is commented. Complexity of Gröbner Basis computations that apply for the chosen parametrization are given in section 8. Finally, the reader can toy with an example built using Mathematica in section 10, where the code is also given.

1.1 Background

Given a polynomial $p(t) = \sum_{i=0}^{n-1} a_0 t^i \in F_q[t]$ and its analogue polynomial $g(t)$ expressed as the product of its roots, so $p(t) = g(t) = \prod_{i=0}^{n-1}(t - x_i),\ x_i \in F_q$ it results that we can express $p(t)$ as $g(t)$: a monic univariate polynomial on the variable t with the first n elementary symmetric polynomials as coefficients. Historically, the roots of a polynomial $p(t)$ have been studied from the viewpoint of the product of its roots. For example, we can relate the coefficients of a polynomial $p(t)$ with the first n elementary symmetric polynomials where the variables are the roots x_i s.t $p(x_i) = 0$. Extending to Vieta's Formulas, this means that the product of the roots of $p(x)$ and the coefficients are strongly related.

2 The MVSYM primitive

From now, denote the polynomial $g(t) - t^d$ as $G(X, t)$ where t is the univariate variable and $X = (x_1, \ldots, x_n)$ are the roots. Note how the monomial t^d has been eliminated, as it won't contribute to the cryptosystem. Following, $G(X, t)$ serves as a trapdoor for hiding the roots of a polynomial $p(x) \in F_q[X]$ since there exists efficient root finding algorithms over fields (and their extension). A natural question is to ask ourselves if $G(X, t)$ could be placed as

the central polynomial map in a MPKC scheme. Let $\overline{G}(X)$ be the map representation of the polynomial $G(X,t)$, where every equation of the map is an elementary symmetric polynomial .

The very first drawback noticed is that a public key derived using this primitive $P(X) = T \circ \overline{G} \circ S(X)$ contains monomials of degree $d \geq 2$, which results in an increase of memory needed for its representation. That has an impact on network transmission as well and in the evaluation of the public key when a plaintext is provided.

Here we find distinct cases for the parametrization of the polynomial $G(X,t)$. Given d, n, m where $d = Deg(G(X,t)) = m$ and n the n$^{\underline{o}}$ of variables:

1. The primary case is when $d = m = n$, which means that the coefficients of $G(X,t)$ are the first n elementary symmetric polynomials in n variables, which are the roots x_i, so $Deg(g_i(X)). = i$ The polynomial $G(X,t)$ has degree $n-1$ since we removed the monic monomial t^n. Consequently, in the primary case we find the trivial degree partition $\lambda = (e_1, \ldots, e_n) = (1, \ldots, 1)$.

2. The secondary case is when $d = m \neq n$. This means that there exists an integer partition $\lambda \vdash d = (e_1, \ldots, e_n)$ with at least one $e_i > 1$ such that $G(X,t) = t^d - \prod_{i=1}^{n}(t - x_i)^{e_i}$. Then $G(X,t)$ has degree $d = 1 - \sum_{i=1}^{n} e_i$ as we removed the monic monomial t^d. The coefficients of $G(X,t)$ are not the n first symmetric polynomials anymore, however $Deg(g_i(X)) = i$. So both cases converge on the degree of each polynomial in $\overline{G}(X)$.

The secondary case is only commented in the cryptosystem in a general way, the rest of the paper focuses in the primary case over an extension field. Something that can be commented of the secondary case is that when adding a linear transformation S, the action of S in the variable set $x = (x_1, \ldots, x_n)$ doesn't alter the degrees of each $g_i(X)$, but it does include new monomials. These condition tells us that the degrees $e_i \in \lambda$ are visible in the multivariate equations computed by $\overline{G} \circ S(X)$ for the secondary case.

3 Description of the Cryptosystem based in MVSYM

Given the number of variables n, the degree d, two (affine) transformations $T \in GL(d,q) \times F_q^d$ $S \in GL(n,q) \times F_q^n$ and the set of variables $x = (x_1, \ldots, x_n) \in F_q$

Remark The reader can head to section 5 and to point 5.1.2 to find the selected parametrization for the cryptosystem. The following description of the cryptosystem is mostly general and differs from the implementation seen in the referenced points.

3.1 Encryption

1. $S(x)$ acts linearly on the set of variables, giving a linear combination tuple $x' = (x'_1, \ldots, x'_n)$.

2. Select an integer partition $\lambda \vdash d = (e_1, \ldots, e_n)$ with length n. The sum of partition elements equals the degree so $d = \sum_{i=1}^{n} e_i$.

3. The primary case has $e_i = 1$ as $n = d$. The secondary case has $n < d$ as there's at least one $e_i \geq 1$.

4. Expand the polynomial $G(X,t) = t^d - \prod_{i=1}^{n}(t - x'_i)^{e_i}$ modulo p or $f(x)$ if we are in an extension field and represent it as a d equation map identified by $\overline{G}(X) = y'$.

5. Obtain the final equation $Ty' = y$ which is $P(x) = y$

3.2 Decryption

The decryption stage is done backwards, where the ciphertext $P(m) = y$ and the digest of the message plus a public salt $H(m + s)$ are given

1. Recover $y' = T^{-1}y$

2. Solve the equation $t^d + G(y', t) = t^d + \sum_{i=1}^{d} y'_i t^{i-1} = 0$ which gives $\overline{G} \circ S(X) = \prod_{i=1}^{n} (t - x'_i)^{e_i}$.

3. As a consequence of root finding not giving the roots in the correct position, we must guess the valid ordering for the sequence of the roots. This is up to $n!$ permutations in the worst case for the primary case as the algorithm works with n variables in total.

4. For every vector x' that we obtained permuting the roots, compute $x = S^{-1}x'$ and verify if $H(M + s) = H(x + s)$

Clearly, we notice that the decryption stage is bounded by the $n^{\underline{o}}$ of permutations applied to x' until we find the correct decryption in x. A natural question is to ask how we can bypass this search for the right permutation. If Bob selects values $x_1 < x_2 < \ldots < x_n$ or the opposite case $x_1 > \ldots > x_n$ it's easy for Alice to recover the right x' from the root finding invocation on $\overline{G} \circ S(X)$ as the ascending or descending order of the roots gives us the right values.

4 Digital Signatures and MVSYM

Digital Signatures are pursued in the field of MPKC. Generally, a bijective Cryptosystem has an analogue Digital Signing algorithm, where the value y is the digest of a salted message. When a cryptosystem is surjective, the condition $P(X) = P(Z) = y$ for multiple pairs (X, Z) is satisfied. As a consequence, it's relevant to classify how many surjections do arise for a particular point-image $P(X) = Y$ in MVSYM.

Furthermore, the inverse of $G(X, t)$ gives it's roots, as these are the elements $X = (x_1, \ldots, x_n)$ but in a permuted order, we obtain $\overline{G} \circ P \circ S(X)$ where P is a permutation matrix, or a Linearised Polynomial that permutes the input polynomial's coefficients. Here, let P be any matrix from the permutation group of matrices that is isomorphic to the symmetric group S_n. Then the obvious case for surjectivity is that any permutation matrix that gives an ordering for the roots, doesn't alter the output of the polynomial map $\overline{G}(X)$. However, there are $n!$ permutations that match the same output for $\overline{G}(X)$, which alerts us that when the number of variable grows, each ciphertext has $n!$ surjections. Thus if q^n is the cardinality of the space of the ciphertext, we end up having $\frac{q^n}{n!}$ possible plaintexts. The same happens after a linear change of variables, only if the transformation S is non-singular.

$$\forall i, j : \overline{G} \circ P_i \circ S(X) = \overline{G} \circ P_j \circ S(X) \quad 1 \leq i < j \leq n! \quad P_i \in Perm(S_n)$$

When working with Digital Signatures over MVSYM, the issues of obtaining the correct sequence X by permuting the roots, as in the decryption stage, is not a problem anymore. Any permuted sequence of $S(X)$ will match the digital signature, so the process offers a considerable speedup when the number of variables grows.

1. Bob sends a server certificate to Alice where $P(X)$ as his public key.

2. Alice validates Bob's certificate.

3. Bob computes $y = H(m||s)$ which is a n-tuple that represents the digest of a salted message

4. He recovers $y' = T^{-1}y$ and find the roots of the equation $t^d + y'(t) = 0$ which gives $x' = \overline{G} \circ P_i \circ S(X) = \prod_{i=1}^{n} (t - x'_i)^{e_i}$.

5. Computes $x = S^{-1}x'$ and verifies if $P(x) = y$

6. Bob sends to Alice the pair (x, y, m, s).

7. As Alice trusts Bob's certificate, she already has $P(X)$ and verifies whether $P(x) = y = H(m||s)$. If it's correct she knows that Bob sent that particular message.

   Now, forgery is possible when an attacker finds a value $P(x') = y' = H(m'||s')$. Then he could impersonate Bob, sending the tuple (x', y', m', s'). The thing is that the attacker must match $P(X) = y$ to a digest of a salted message, which is as hard as inverting $P(X)$ to obtain the point x that matches $y = H(m'||s')$. The recommended parametrization will be further commented.

5 Analyzed Parametrization

Parametrization is an important step to evaluate the computational needs for the cryptosystem. Here, the focus is set on the case $m = n$ where every variable x_i is in the extension $GF(p^n)$. The cases that set q as a prime field or those based in the secondary case $m > n$ where multi-degree parameters appear ($e_i > 1$) are not covered here.

As the reader can see, we're focusing in a particular subsinstance of MVSYM, thereafter, two vital cases for parametrization using a finite field extension are found: the case when $m = n = 2$ and $m = n = 3$. This paper focus on $m = n = 2$,and doesn't work with cubic cases.

The most important thing to recall during the reading is that polynomial equations in the system are taken from $F_{p^n}^2$ to F_p^{2n}.

5.1 Public Key Generation on the quadratic case over F_2

When $m = n = 2$ we find that $G(X, t) = t^2 + (x_1 + x_2)t + x_1x_2$, thus by observation, the monomial x_1x_2 and $x_1 + x_2$ where $x_1, x_2 \in F_{2^{n_v}}$ gives an oil vinegar polynomial as x_1 and x_2 provide distinct variable sets. Oil-Vinegar has been extended studied, in the case $o = v$ it can be broken by the cryptanalysis given in [1] . The paper focuses on retrieving an invariant subspace of the Oil space as the structure present in the quadratic forms allows the cryptographer to mount such attack, even in characteristic 2. Besides, it's commented in [2] section 14 that when $2o \leq v \leq \frac{o^2}{2}$ the scheme remains secure against the KS attack for OV. In literature, such condition is called Unbalanced Oi-Vinegar. Let's document the general case that applies to this parametrization of the scheme. First let $n = n_o + n_v$ be the number of variables the field $q = GF(p^{n_v})$ and the irreducible polynomial $f(t)$:

$$\varphi(\overrightarrow{x}) : F_q^n \to F_{q^n} \quad (a_1, \ldots, a_n) \mapsto \sum_{i=0}^{n-1} a_i t^i$$

$$\overrightarrow{x_1} = (x_1, \cdots, x_{n_o}, \overbrace{0, \ldots, 0}^{n_v - n_o}), \overrightarrow{x_2} = (x_{n_o+1}, \ldots, x_{n_v})$$

$$\varphi(x_1) + \varphi(x_2)) = \sum_{i=1}^{n_o} x_i t^{i-1} + \sum_{i=1}^{n_v} x_{n_o+i} t^{i-1} \pmod{f(t)}$$

$$\varphi(x_1)\varphi(x_2) = \sum_{i=1}^{n_o} x_i t^{i-1} \cdot \sum_{i=1}^{n_v} x_{n_o+i} t^{i-1} \pmod{f(t)}$$

After the reduction of both polynomial equations modulo $f(t)$, take the polynomials as vectors and join them. We end up having $2n_v$ equations on $n_o + n_v$ variables. From now, consider the quadratic forms of this polynomial map:

$$\varphi^{-1}(\varphi(x_1) + \varphi(x_2))) = (f_1(x_1, \ldots, x_n), \ldots, f_{n_v}(x_1, \ldots, x_n)) = (x^T Q_1 x, \ldots, x^T Q_{n_v} x)$$

$$\varphi^{-1}(\varphi(x_1 x_2)) = (f_{n_v+1}(x_1, \ldots, x_n), \ldots, f_{2n_v}(x_1, \ldots, x_n)) = (x^T Q_{n_v+1} x, \ldots, x^T Q_{2n_v} x)$$

After the obtention of both polynomials, select the pair of transformations $T \in F_2^{2n_v \times 2n_v}, S \in F_2^{n \times n}$ and compute each polynomial $p_i(X)$ of the public key $P(X)$ as follows:

$$p_i(x_1, \ldots, x_n) = x^T S^T (\sum_{j=1}^{2n_v} t_{i,j} Q_j) S x$$

Before applying T, notice that the first half of the quadratic forms represent the linear mapping $x_1 + x_2$, as we are in characteristic 2, linear forms can be stored in a quadratic forms too, i.e: $x_1^2 + x_3^2 + x_5^2 = x_1 + x_3 + x_5$ can be stored in a diagonal matrix. This means that the action of T in this quadratic form returns a linear combination of the $S^T Q_i S$ where the Q_i's in the first half are diagonal matrices.

The main diference of Oil-Vinegar schemes with MVSYM for the unbalanced case is that MVSYM has $2n_v$ polynomials on $n_o + n_v$ variables, so $m > n$ but in UOV we have $m < n$. In addition, the inversion of the primitive MVSYM depends entirely on root finding over F_q, not in Gaussian Elimination once the vinegar values are substituted.

5.1.1 KS Attack on Balanced Oil-Vinegar

If we set up $n_v = n_o$ and $n = n_o + n_v$ then this scheme raises a distinct structure on the final quadratic form system found in $P(X)$, concretely we've $m = n = 2n$ equations-variables.

The structure of a matrix that represents a quadratic form of the Balanced Oil-Vinegar map must have the structure presented here [1]. The characteristic 2 is covered as well, as this is the case. every Q_i is an upper triangular matrix such that:

$$\forall i : 1 \leq i \leq n \quad f_i(X) = X^T Q_i X = \left[x_1 \dots x_{n/2}, x_{n/2+1}, \dots, x_n\right] D_i \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ x_{n+1} \\ \vdots \\ x_{2n} \end{bmatrix}$$

$$\forall i : n+1 \leq i \leq 2n \quad f_i(X) = X^T Q_i X = \left[x_1 \dots x_{n/2}, x_{n/2+1}, \dots, x_n\right] \begin{bmatrix} 0 & A_{1,i} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{n/2} \\ x_{n/2+1} \\ \vdots \\ x_n \end{bmatrix}$$

Consequently, the conditions of section 3 in [1] seem not to apply to the quadratic forms in $P(X) = (x^T M_1 x, \dots, x^T M_{2n} x)$ since the diagonal matrices D_i change the structure of the matrices $G_j^{-1} G_i$. It remains open if a modification of the KS technique will break this balanced OV map.

5.1.2 Recommended Case: Unbalanced Oil-Vinegar

In the other hand, the quadratic set can be strengthen by including the condition [2] $2o \leq v \leq \frac{o^2}{2}$. Here, the both variables x_1, x_2 are taken over the bigger field $GF(2^{n_v})$ as $n_v > n_o$. Then we have $2n_v$ equations in $n = n_o + n_v$ variables.

Setting $n_v = 2n_o$ the rule to calculate the n$^{\underline{o}}$ of equations m and the n$^{\underline{o}}$ of variables n is $m = 2n_v = 4n_o, \quad n = n_o + n_v = 3n_o$. These properties are useful in section 8 for estimating the effective number of operations required for a Gröbner Basis computation.

6 Public Key Size

In the field of Multivariate Cryptography, the public keys found in schemes are typically quadratic sets over a prime characteristic. This means that every quadratic polynomial $p_i(x)$ having degree 2 is representable by the equation $x^T Q_i x$ where Q_i is the square matrix over F_q associated to the quadratic form. However, the scheme introduced in this paper, in most of the cases cannot be represented as a system of quadratic forms. section 5 illustrates why $m = 2n_v$ and the parametrization $n_o = 64, n_v = 128, m = 256, n = 192$ is selected in point 5.1.1.

Moreover, the space complexity of a quadratic system of m n-variate polynomials over F_p is $\mathcal{O}(mn^2 \log_2(q))$ and $\mathcal{O}(n^3 \log_2(q))$ when $m = n$.

Complexity for fields over characteristic 2 When the field $q = p^k$ has characteristic 2, we've to distinguish between the Balanced Oil-Vinegar and Unbalanced cases.

Balanced Oil-Vinegar The bound is

$$\mathcal{O}(2n \cdot (2n)^2 \log_2(p)) = \mathcal{O}(8n^3)$$

as we work over the extension F_{2^n} we have $m' = n' = 2$ thus 2 equations and variables. Translating it to F_2 we end up having $2n$ equations on $2n$ variables. For example $m' = n' = 2, n = 64, q = F_{2^{64}}$ so $\mathcal{O}(8 \cdot 64^3) \approx 256KB$ for the storage needed for the quadratic forms in 128 equation and variables of the polynomials in $P(X)$.

Unbalanced Oil-Vinegar As seen in subsection 5.1 we end up having $2n_v$ equations in $n_o + n_v$ variables. Then the bound is

$$\mathcal{O}(2n_v(n_o + n_v)^2)$$

For example, with the condition $n_v = 2n_o$ set up $n_o = 64, n_v = 128$ and $n = n_o + n_v = 192$. The storage needed is calculated as $\mathcal{O}(2 \cdot 128 \cdot 192^2) = 1.125MB$ for 256 equations in 192 variables. Another example with $n_o = 32, n_v = 64, n = 96$ then $\mathcal{O}(2 \cdot 64 \cdot 96^2) = 144KB$ for 128 equations in 96 variables, however the last parametrization seems not to be secure as found in section 8.

Other cases Note that the case $m = n = d$ is called the primary case found in section 2 which has a trivial partition λ with every degree being 1. If we fix $n = 2$ and $m = d > n$ we end up having a non trivial partition λ that still sums to d however the monomial terms involved in the polynomial expression either on F_q or taken into F_p are not longer quadratic.

In the other hand, if we stick to the primary case with the condition $m = n > 2$, the public key $P(X)$ taken from F_q to F_p doesn't give full quadratic polynomials anymore thus we must stick to the complexities for the non-quadratic case.

7 Isomorphism of Polynomials

The Isomorphism of Polynomials Problem was introduced by [3] in an attempt to add a layer of security to MPKC schemes. In this scheme, the central polynomial is publicly known, which is the polynomial $G(X, t)$ which is a polynomial having the first d elementary symmetric polynomials as its coefficients.

If we only consider the IP1 assumption [4], and set $P(X) = \overline{G} \circ S(X)$, it results that $P(X)$ is reversible and S can be entirely determined as we know how to invert the central map $G(X, t)$. First, set $x = e_i$ then $P(e_i) = \overline{G} \circ S(e_i)$ thus we compute the i-th column of S by $Se_i = \overline{G}^{-1}(Se_i)$. This is equal to set $t^d + G(Se_i, t) = 0$ and apply root finding over p or F_q depending on which field we're working. Each root finding gives us Se_i which we must reorder up to $n!$ permutations. Then by linearity $S(e_i + e_j) = S(e_i) + S(e_j)$ thus we can check in the list of column candidates those that satisfies these relations until we build the complete transformation S. This process can be found in [5] section 6.2.

Another important remark, is when $P(X) = T \circ \overline{G}(X)$. In literature, this kind of expression is not considered into the IP1 assumption. It results that any scheme presented in this form is broken, where $G(X, t)$ and $\overline{G}(x)$ are publicly known along with the inversion method. To prove that fix, x_i so that $\overline{G}(x_i) = e_i$ then compute $P(x_i) = T \circ \overline{G}(x_i) = T(e_i)$ thus recovering the i-th column of T. Obviously, we need to compute $t^d + \overline{G}^{-1}(e_i) = x_i$ to retrieve the correct value x_i that is sent to the canonical vector e_i under the image of $\overline{G}(X)$.

Then the only conclusion that comes up to make $P(X)$ secure to these kinds of attacks is to base the scheme in the IP2 assumption, thus the central map is in between of two invertible transformations T, S such that $P(X) = T \circ \overline{G} \circ S(X)$.

8 Gröbner Basis

In MPKC, algorithms for the computation of a Gröbner Basis of the Ideal $< f_1, \ldots, f_n >$ are extensively studied. Specially those that involve dense linear algebra to build the Macaulay matrix $M_{n,D}$. In the past years, a lot of effort has been put for estimating the degree of regularity d_{reg} of a polynomial system of equations over F_q. We know that it can be obtained by calculating the coefficients of the Hilbert Series [6] $S_{m,n}$ and taking the degree of the first non-positive coefficient when the system is semi-regular, which is the case we study.

Continuing with the analysis, It results that the total n$^{\underline{o}}$ of operations to come up with a Gröbner Basis is bounded by this parameter. However, despite of asymptotic bounds being a necessary tool for cryptanalysis, it doesn't mean that it should cover the complexity found in schemes for example, based in the HFE [8]. The most popular algorithms that are used to compute such a basis are F_4, F_5, XL. In particular, F_4, F_5 require a number of operations that depends on the size of the Macaulay matrix $M_{n,d_{reg}}$ and the linear algebra constant w. The natural conclusion is to focus on the general complexity and asymptotics found in [6] [7].

First, consider the the ideal I generated by $< f_1, \ldots, f_m, x_1^q - x_1, \ldots, x_n^q - x_n >$ over a field of characteristic 2, which corresponds to the Frobenious Criterion as seen in [7], section 4.1, and results in adding the equations $x_i^q = x_i$ to the system so we end up having $m + n$ equations on n variables.

The parametrization given in subsection 5.1 has $2n_v$ equations on $n_o + n_v$ variables. After the adding the equations $x_i^2 - x_i = 0$, we obtain $2n_v + n_o + n_v$ equations in $n_o + n_v$ variables. Since $n_v = 2n_o$, this gives $m = 2n_v + n_o + n_v = 4n_o + n_o + 2n_o = 7n_o$ and $n = n_o + n_v = n_o + 2n_o = 3n_o$, thus clearly $\frac{m}{n} = \frac{7}{3}$ which tends to a constant. If we don't take into account the Frobenius Criterion the constant tends to $\frac{4}{3}$. The asymptotic for the case where $N = \frac{m}{n}$ tends to a constant is found in section 6 in [7], which applies to this case as $N = \frac{7}{3} > \frac{1}{4}$ is:

$$\frac{d_{reg}}{n} = \frac{1}{2} - N + \frac{1}{2}\sqrt{2N^2 - 10N - 1 + 2(N+2)\sqrt{N(N+2)}} + o(1)$$

Plugging $N = \frac{7}{3}$ and $n = n_o + n_v = 64 + 128 = 192$, the approximation 0.0450835 is obtained, when multiplied by 192 we obtain an approximation

$$d_{reg} \approx 8.65603 \approx 8$$

. In [6], page 9 the bound for the nº of operations required by F_5 given a semi-regular polynomial set over F_2 is about

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

Then as $m = 7n_o \quad n = 3n_o \quad m = 7 \cdot 64 = 442 \quad n = 3 \cdot 64 = 192$ and computing a Gröbner basis by the F_5 algorithm requires $\approx 2^{125}$ operations.

In the other hand, [7] offers a continuity for the approximation to the nº of operations required to compute a Gröbner Basis.

As we computed $D_0 = \frac{d_{reg}}{n}$ then

$$D_1 = -(1 - D_0)\log_2(1 - D_0) - D_0 \log_2 D_0$$

and the global cost of the basis computation is $(2^{D_1})^{nw}$. For example, in the previous calculation we found that $D_0 = \frac{d_{reg}}{n} = 0.0450835$, then $D_1 \approx 0.265133$, so $(2^{D_1})^{nw} \approx 2^{127}$.

An interesting case is when $n_o = 32$ so $n_v = 64$ and $m = 128, n = 96$ over F_2. Then for the algorithm F_5 to work we end up having $m = 7 \cdot 32$ $n = 3 \cdot 32$. Both aforementioned complexities converge to $\approx 2^{63}$ operations, thus both formulas are useful when estimating.

From here we can *possibly* conclude that the parametrization $n_v = 2n_o$ must satisfy $n_o \geq 64$ if we want to achieve minimal security

9 Improving Public Key Generation: Linear Algebra and the polynomial remainder in Finite Fields

If we select the parametrization $m' = n' = 2$ and $q = F_2^{n_v}$ and we end up having a quadratic system of $2n_v$ equations and $n = n_o + n_v$ variables. However, the steps taken to produce the map $\overline{G}(X)$ involve symbolic polynomial multiplication and transformations under symbolic variables.

Recall that $(x_1', x_2') = S(x_1, x_2)$ so $t^2 + (x_1' + x_2')t + x_1'x_2'$ where $x_1, x_2 \in GF(2^k)$. Then the hardest symbolic computation will happen in the monomial product $x_1'x_2'$. Both monomials contain a linear combination of the columns of $S \in F_{2^{n_v}}^{2\times 2}$ and the vector $(x_1, x_2) \in F_{2^{n_v}}^2$. This process is slow, i.e when the extension length n_v grows. However, we can use linear algebra to represent the remainder of a polynomial multiplication of two elements in $GF(2^{n_v})$ by a $n_v \times n$ matrix over F_p.

If we multiply two *symbolic* polynomials $p(x), q(x) \in GF(2^{n_v})$, where $Deg(p(x)) = n_o - 1$ and $Deg(q(x)) = n_v - 1$ their product results in a polynomial of degree $n - 1$ and the reduction in the field will give a polynomial of degree $n_v - 1$, this is a n_v tuple.

1. The following algorithm computes the quadratic map of the product $p(x)q(x) = r(x) = \sum_{i=0}^{n_o-1} a_i x^i \cdot \sum_{i=0}^{n_v-1} b_i x^i = \sum_{i=0}^{n-1} c_i x^i$ in the univariate polynomial ring $Z_p[X]$. Using basic arithmetic on the set of variables where $n_v = 2n_o$ and $n = n_o + n_v$.

Input: The triplet (n, n_o, n_v)
Output: The *pyramidal* multiplication map of $\overline{r}$ from the product $p(x)q(x)$ having n quadratic equations
Function *pyramUOV(start, ord)* **is**
map$_1 \leftarrow$ List()
map$_2 \leftarrow$ List()
for $i = 1, i \leq n_v, i++$ **do**
pv $\leftarrow 1$
poly $\leftarrow 0$
for $j = i, j > 0, j--$ **do**
poly $\leftarrow poly + x_{pv}x_{n_o+j}$
if $pv == n_o$ **then**
Break
else
pv++
end
end
Insert(map1,poly)
end
for $i = 1, i \leq n_o, i++$ **do**
pv $\leftarrow n_o$
poly $\leftarrow 0$
for $j = i, j > 0, j--$ **do**
poly $\leftarrow poly + x_{pv}x_{3n_o-j+1}$
pv–
end
Insert(map2, poly)
end
return Join(map$_1$,Reverse(map$_2$))
end

2. Now let $\overline{r_f}$: the reduction of $\overline{r}$ as a polynomial in $GF(2^{n_v})$, which is converted back to a tuple in $F_2^{n_v}$.

3. Then $\overline{r_f}$ is the quadratic map corresponding to the monomial x_1x_2 in $GF(2^{n_v})$

4. To obtain $\overline{G}(X)$, join $\overline{r_f}$ with the map corresponding to the sum of polynomials $x_1 + x_2$ in $GF(2^{n_v})$.

However, in step 3 we are bounded by the time that takes to compute the polynomial remainder of an univariate polynomial having quadratic monomials as coefficients, which is costly for example when $n_o = 64, n_v = 128$. It results that the polynomial remainder operation can be translated to Linear Algebra, as reducing a $n-1$ degree polynomial modulo $f(x)$ would yield a $n_v - 1$ degree polynomial. By simple inspection, derive the following relation:

$$\begin{bmatrix} r_{1,1} \dots r_{1,n} \\ \dots\dots\dots \\ \dots\dots\dots \\ r_{n_v,1} \dots r_{n_v,n} \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ \vdots \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} d_1 = \sum_j^n r_{1,j}c_j \\ \vdots \\ d_{n_v} = \sum_j^n r_{n_v,j}c_j \end{bmatrix}$$

Thereafter, we can read the values $r_{i,j}$ from each d_i which gives the $n_v \times n$ transformation matrix R that sends the product $p(x)q(x) = r(x)$ in $Z_p[x]$ to it's reduction in F_q. Why? Taking $(c_1, \dots, c_n)$ as a polynomial and reducing it modulo $f(x)$ gives a polynomial in $GF(2^{n_v})$ having linear combinations of the c_i as coefficients. Taking such polynomial as a map gives the values $(d_1, \dots, d_{n_v})$.

After that, obtain the Quadratic Forms of every polynomial $g_i(X) \in \overline{G}(X) \quad \forall 1 \leq i \leq n_v$ And compute $P(X)$ as

$$p_i(x_1, \ldots, x_n) = x^T S^T (\sum_{j=1}^{2n_v} t_{i,j} Q_j) S x$$

For example, using the algorithms shown in section 10 for the parametrization $n_o = 64, n_v = 128, m = 256, n = 192$ it takes ≈ 40 seconds, which is the recommended setup.

10 Toy Example in Mathematica

The research made has been experimentaly tested on Mathematica and Sage. The drawback of using Mathematica for Finite Field arithmetic is that it lacks of root finding and factorisation methods over finite field extensions. Sage supports these methods, so for toying with some examples, the public key generation happens in Mathematica, and the decryption is an hybrid approach using Mathematica and Sage. The same routines can be written on C/C++ with the aid of NTL for the extension field arithmetic and linear algebra.

Moreover, the code listed here, generates a public key for the case $m = n = 2$ and $q = GF(2^{n_v})$. So we end up having $2n_v$ equations on $n_o + n_v = n$ variables. The Public Key's generating algorithm computes the multiplication map of two polynomials in $Z_2[X]$ first, then reduces it by $f(x)$ using Linear Algebra, yielding n_v equations in n variables. After that, quadratic forms from these quadratic monomials are extracted. Then the sum map $x_1 + x_2$ of polynomials is computed because a linear map is represented as a quadratic form by the condition $x_i^2 = x_i$ in characteristic 2. Finally, joining the sum map and the multiplication map gives the map $\overline{G}(X)$, which is composed of $2n_v$ quadratic equations in $n = n_o + n_v$ variables and equations. The rest is to compute the public key by using Linear Algebra on the quadratic forms from the central polynomial map $\overline{G}(X)$.

In[1]:=
```
GenRndMat[p_,n_]:=(
        mat=0;
        rnk=0;
        While[rnk < n,
                mat = RandomInteger[{0,p-1},{n,n}];
                rnk=MatrixRank[mat,Modulus->p];
        ];
        Return[mat];
)
```

In[2]:=
```
pyramuov[n_, no_, nv_] := (
        syms = Table[
        Symbol[StringJoin[ToString[x], ToString[i]]], {i, 1, 3*no}];
        j = 1;
        map1 = List[];
        map2 = List[];
        For[i = 1, i <= nv, i++,
                pivot = 1;
                poly = 0;
                For[j = i, j > 0, j--,
                        poly += syms[[pivot]]*syms[[no + j]];
                        If[pivot == no,
                                Break[];
                                ,
                                pivot++;
                        ];
                ];
                map1 = Insert[map1, poly, Length[map1] + 1];
        ];
        For[i = 1, i < no, i++,
                pivot = no;
                poly = 0;
                For[j = i, j > 0, j--,
                        poly += syms[[pivot]]*syms[[3*no - j + 1]];
                        pivot--;
                ];
                map2 = Insert[map2, poly, Length[map2] + 1];
        ];
        mulmap = Join[map1, Reverse[map2]];
)
```

In[3]:=
```
GenQF[qset_, no_, nv_] := (
        Qs = Array[0 &, Length[qset]];
        o = Take[symsE, {1, no}];
        v = Take[symsE, {no + 1, no + nv}];
        For[i = 1, i <= Length[qset], i++,
                mons = MonomialList[qset[[i]], o];
                mons = Table [mons[[l]]/symsE[[l]], {l, 1, Length[mons]}];
                qmat = ConstantArray[0, {no + nv, nv + no}];
                For[j = 1, j <= Length[mons], j++,
                        monscoeff = MonomialList[mons[[j]], v];
                        For[k = 1, k <= Length[monscoeff], k++,
                                pos = Position[v, monscoeff[[k]]];
                                If[pos != {},
                                        pos = no + pos[[1]][[1]];
                                        qmat[[j]][[pos]] = 1;
                                ];
                        ];
                ];
                Qs[[i]] = qmat;
        ];
        Return[Qs];
        )
```

```
In[4]:= GenLF[lset_] := (
            Qs = Array[0 &, Length[lset]];
            For[i = 1, i <= Length[lset], i++,
                    mons = MonomialList[lset[[i]]];
                    qmat = ConstantArray[0, {2*Length[lset], 2*Length[lset]}];
                    For[j = 1, j <= Length[mons], j++,
                            pos = Position[symsE, mons[[j]]];
                            If[pos != {},
                                    pos = pos[[1]][[1]];
                                    qmat[[pos]][[pos]] = 1;
                            ];
                    ];
                    Qs[[i]] = qmat;
            ];
            Return[Qs];
    )

In[5]:= PolyMul[p_, n_, no_] := (
            irr = IrreduciblePolynomial[t, p, n - no];
            pyramuov[n, no, n - no]; symsred =
            Table[Symbol[StringJoin[ToString[r], ToString[i]]], {i, 1,
                    n - 1}]; matred =
            CoefficientArrays[
            CoefficientList[
            PolynomialMod[symsred.t^Range[0, n - 2], irr, Modulus -> p], t,
            n - no], symsred][[2]] // Normal; mulmapF =
            PolynomialMod[matred.mulmap, aa, Modulus -> p];
            Return[mulmapF];
    )

In[6]:= GenSysE[p_, n_, no_, nv_] := (
            symsE = Table[Symbol[StringJoin[ToString[x], ToString[i]]], {i, 1, n}];
            mulmapF = PolyMul[p, n, no];
            qf = GenQF[mulmapF, no, nv];
            lf = GenLF[Join[o, Array[0 &, nv - no]] + v, no + nv];
            Qs = Join[lf, qf];
            S = GenRndMat[p, n];
            T = GenRndMat[p, Length[Qs]];
            QsS = PolynomialMod[Table[Transpose[S].Qs[[i]].S, {i, 1, Length[Qs]}],
            aa, Modulus -> p];
            TQsS = PolynomialMod[T.QsS, aa, Modulus -> p];
    )

In[7]:= Encipher[plaintext_]:=(
            Table[Mod[plaintext.TQsS[[i]].plaintext, 2],
             {i, 1, Length[plaintext]}] // Return;
    )
```

1. Generate a system of equations for the case $n_o = 3, n_v = 6, m = 12, n = 9$. So GenSysE[2,9,3,6] The plaintext is $X = (x_1, \ldots, x_9)$ where the variables are taken in the field $GF(2^{n_v}) = GF(2^6)$.

2. Call Encipher passing a $n = 9$ bit tuple as the plaintext argument. It will return a $2n_v = m = 12$ bit tuple, this is the ciphertext.

3. Apply $c' = T^{-1}c$ where c is the tuple representing the ciphertext.

4. Solve the equation $t^2 + G(c', t) = 0$ over $GF(2^6)$ in Sage to retrieve $x'_1, x'_2 \in GF(2^6)$. Over F_2 both polynomial tuples are seen as $x'_1 = (x_{1,1}, \ldots, x_{1,3})$ and $x'_2 = (x_{2,1}, \ldots, x'_{2,6})$.

5. Apply $S^{-1}(x'_{1,1}, \ldots, x'_{1,3}, x'_{2,1}, \ldots, x'_{2,6}) = (x_1, \ldots, x_9)$.

Let's give a detailed explanation of the whole process. For example, let $P(X)$:

x1x3 + x1x5 + x1x7 + x1x8 + x1x9 + x1 + x2x4 + x2x7 + x2x8 + x2 + x3x4 + x3x5 + x3x6 + x3x9 + x4x5 + x4x7 + x4x8 + x5x6 + x5x7 + x5x9 + x5 + x6x7 + x6x8 + x6x9 + x7 + x8
x1x3 + x1x7 + x1x9 + x2 + x3x4 + x3x5 + x3x8 + x3x9 + x4x6 + x4 + x5x7 + x5x9 + x5 + x6x7 + x6 + x8x9 + x9
x1x2 + x1x4 + x1x8 + x1x9 + x1 + x2x4 + x2x5 + x2x6 + x2x8 + x2 + x3x4 + x3x5 + x3x6 + x3x7 + x4x5 + x4x6 + x4x7 + x4x8 + x4x9 + x5x7 + x5 + x6x7 + x7x8 + x8
x1x3 + x1x4 + x1x5 + x1x6 + x1 + x2x3 + x2x7 + x2x8 + x2x9 + x3x4 + x3x5 + x3x6 + x3x7 + x3x8 + x4x5 + x4x7 + x4x9 + x5x7 + x6x8 + x7x8 + x7 + x8x9 + x8 + x9
x1x2 + x1x3 + x1x4 + x1x8 + x1x9 + x1 + x2x4 + x2x5 + x2x6 + x2x8 + x2 + x3x8 + x3 + x4x5 + x4x6 + x4x7 + x4x9 + x4 + x5x7 + x5 + x6x7 + x7x8 + x7x9
x1x2 + x1x8 + x1x9 + x1 + x2x4 + x2x5 + x2x6 + x2x8 + x3x8 + x4x7 + x4x8 + x4 + x5x8 + x5x9 + x5 + x6x8 + x6x9 + x7x9 + x8x9
x1x5 + x1x7 + x1x8 + x1x9 + x2x4 + x2x7 + x2x8 + x3x7 + x3x8 + x3x9 + x4x5 + x4x7 + x5x6 + x5x7 + x5x9 + x6x7 + x6x8 + x6x9 + x7x9 + x7 + x8 + x9
x1x5 + x1x8 + x2x4 + x2x7 + x2x8 + x2 + x3x6 + x3x8 + x4x5 + x4x6 + x4x7 + x4x8 + x5x6 + x6x8 + x6x9 + x7 + x8x9 + x9
x1x2 + x1x5 + x1x6 + x1x8 + x1x9 + x1 + x2x3 + x2x4 + x2x5 + x2x6 + x2x7 + x2x9 + x3x4 + x3x5 + x3x6 + x3x7 + x3 + x4x6 + x5 + x6x7 + x6x8 + x6 + x7x9 + x8x9 + x8 + x9
x1x4 + x1x5 + x1x8 + x2x4 + x2x7 + x2x8 + x3x4 + x3x5 + x3x7 + x3 + x4x7 + x4x8 + x4x9 + x5x6 + x5x7 + x5x8 + x5x9 + x5 + x6x7 + x6 + x7x8 + x7x9 + x9
x1x3 + x2 + x3x4 + x3x5 + x3x6 + x3x7 + x3x8 + x4x8 + x4 + x5 + x7x9
x1x2 + x1x3 + x1x7 + x1x8 + x2x4 + x2x5 + x2x6 + x2x8 + x3x4 + x3x5 + x3x9 + x4x6 + x4x7 + x4x8 + x5x7 + x5x8 + x5 + x6x7 + x6x8 + x6x9 + x7x9 + x8 + x9

Check that in fact $P(X)$ has $m = 2n_v = 12$ equations on $n = n_o + n_v = 3 + 6 = 9$ variables.

1. Every n tuple is decomposed as two polynomials one of degree $n_o - 1$ and the other $n_v - 1$.

2. Let the plaintext be $x = (1, 1, 1, 0, 1, 0, 0, 1, 1)$ a $n = 9$ variable tuple such that it corresponds to the concatenation of two polynomials $x_1 = 1 + a + a^2,\ x_2 = a + a^4 + a^5$ both in $GF(2^{n_v}) = GF(2^6)$.

3. Call Encipher to retrieve the $m = 12$ bit tuple corresponding to the ciphertext $c = (0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1)$.

4. Compute $c' = T^{-1}c = (0, 1, 1, 1, 0, 0, 1, 1)$. The first n_v values from the ciphertext correspond to the operation $x_1 + x_2$. The second half contains the operation x_1x_2. Both operations are taken in $GF(2^6)$.

5. Build the polynomial $G(X, t) = t^2 + \phi(c')$ where $\phi(c')$ lifts the tuple c' from $F_2^{2n_v}$ to $F_{2^{n_v}}^2$, so we end up having two polynomial coefficients in $GF(2^6)$. The polynomial taken in the univariate variable t is $G(t) = t^2 + (1 + a^5) * t + (1 + a + a^2 + a^4 + a^5)$.

6. Sage gives roots $x'_1 = a^2, x'_2 = a^5 + a^2 + 1$, over $GF(2^6)$. The mathematical construction of the algorithm provides that $Deg(x_1) \leq Deg(x'_1) < n_o$.

7. Build the tuple x' of length n, where the first n_o values correspond to the coefficients of x'_1. The rest values are the n_v values of the coefficients of x'_2. Then $x' = (0, 0, 1, 1, 0, 1, 0, 0, 1)$

8. Compute $x = S^{-1}x' = (1, 1, 1, 0, 1, 0, 0, 1, 1)$ which gives the correct plaintext.

References

[1] Aviad Kipnis. Adi Shamir *Cryptanalysis of the oil and vinegar signature scheme* https://link.springer.com/chapter/10.1007%2FBFb0055733

[2] Aviad Kipnis. Jacques Patarin. Louis Goubin *Unbalanced Oil and Vinegar Signature Schemes* https://link.springer.com/chapter/10.1007/3-540-48910-X_15

[3] Jacques Patarin. *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms* https://link.springer.com/chapter/10.1007/3-540-68339-9_4

[4] Jacques Patarin. Louis Goubin. *Improved Algorithms for Isomorphisms of Polynomials* http://www.minrank.org/ip6long.ps

[5] Françoise Levy-Dit-Vehel. Ludovic Perret *Polynomial equivalence problems and applications to multivariate cryptosystems* https://hal.inria.fr/inria-00071464/document

[6] M.Bardet J.-C.Faugère B. Salvy B-Y.Yang *Asymptotic Behaviour of the Index of Regularity of Quadratic Semi-Regular Polynomial Systems* https://www-polsys.lip6.fr/~jcf/Papers/BFS05.pdf

[7] Magali Bardet, Jean-Charles Faugère, Bruno Salvy *Complexity of Gröbner basis computation for Semi-regular Overdetermined sequences over F_2 with solutions in F_2* https://hal.inria.fr/inria-00071534/document

[8] Jean-Charles Faugère and Antoine Joux *Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases* https://hal.inria.fr/inria-00071849/document