Efficient Public Key Generation for Multivariate Cryptosystems

Christopher Wolf
K.U.Leuven ESAT-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
Christopher.Wolf@esat.kuleuven.ac.be or chris@Christopher-Wolf.de
http://www.esat.kuleuven.ac.be/cosic/

Abstract

Asymmetric cryptographic systems using multivariate polynomials over finite fields have been proposed several times since the 1980s. Although some of them have been successfully broken, the area is still vital and promises interesting algorithms with low computational costs, short message, and signature sizes.

In this paper, we present two novel strategies "base transformation" and "adapted evaluation" for the generation of the public key in such schemes. We demonstrate both at the example of the Hidden Field Equations (HFE) system and outline how they can be adapted to similar systems.

In addition, we compare the running time of the previously known technique "polynomial interpolation" with our new developments both from a theoretical perspective and by empirical studies. These experiments confirm our theoretical studies, namely, base transformation is faster than polynomial interpolation. Especially the first step is $O(n^2)$ while it is $O(n^4)$ for polynomial interpolation where n denotes the number of variables. Moreover, the running time of polynomial interpolation is approximately 30% higher than for base transformation.

1 Introduction

During the past years, several asymmetric cryptographic systems were proposed based on multivariate cryptography [MI88, Pat96a, Pat96b, CGP02a, CGP02b]. There are systems both for signing and encryption. The general idea of these systems is very similar: using polynomials over finite fields of different size, they exploit the intractability of the "MQ problem", i.e., multivariate quadratic equations over finite fields are difficult to solve [GJ79, p. 251]. Although some of these systems have been successfully broken (e.g., [Pat95, Fau02, CDF03]), the area is still vital and promises interesting algorithms with rather low computational costs. Moreover, they also allow rather short signature sizes (e.g., only 128 bits for Quartz [CGP02a]). For an encryption algorithm based on HFE, we can expect similar short messages [Pat96b]. This would be of interest for the transmission of session keys without overhead (e.g., for equivalent security, we would have more than 800 bits overhead for an RSA system).

Although several systems were proposed, the question of efficient public key generation did not receive much attention yet. For example, [Pat96b, CGP02a, CGP02b] completely ignore it. However, [MI88] gives a quite general algorithm which can be adapted easily for different systems. In a nutshell, their algorithm uses polynomial interpolation for multivariate polynomials. See Section 3.1 for a detailed description.

In this paper, we present a novel strategy called "base transformation" which deals with the problem of generating the public key for a given private key. It can be seen as a generalisation of the technique sketched in [Pat96a] for generating the public key. We demonstrate "base transformation" for the "Hidden Field Equations" (HFE) system [Pat96b] and outline how it can be adapted to other systems which base their public key on multivariate polynomials over finite fields. Moreover, we show how "base transformation" can be combined with polynomial

interpolation to obtain the "adapted interpolation" technique. In addition, we study the running time of all techniques from a theoretical point of view and use runtime measurements to verify our estimations.

This paper is organised as follows: in Section 2 we give a concise overview of the HFE system. In the third section we describe three different strategies for public key generation, namely "polynomial interpolation", "base transformation", and "adapted interpolation". A toy example of the base transformation technique can be found in the Appendix. Moreover, we compare the running time of these three algorithms from a theoretical point of view. Section 4 presents a speed comparison in an actual implementation. In Section 5 we sketch how the base transformation technique can be adapted to variations of HFE and also other systems. The paper concludes with Section 6.

2 Description of the HFE system

This section gives a concise description of the "Hidden Field Equations" system. Consider a finite field $\mathbb{F}$ with $q = |\mathbb{F}|$ elements, characteristic char $\mathbb{F}$ (a prime), and an extension field $\mathbb{E}$ with degree n over $\mathbb{F}$. The extension field is generated by the irreducible polynomial $i(t)$ over $\mathbb{F}$. This polynomial $i(t)$ has degree $n := \partial i(t)$. For our purpose, we identify this extension field $\mathbb{E}$ with the vector space $\mathbb{F}^n$ as some operations of HFE are not performed in the field $\mathbb{E}$ but in the vector space $\mathbb{F}^n$. This means that every $e \in \mathbb{E}$ can be seen as a vector $e = (f_1, \ldots, f_n) : f_i \in \mathbb{F}$ and, also, as a polynomial in $\mathbb{F}[t]/i(t)$ of degree at most $n-1$. In $\mathbb{E}$, addition is normal polynomial addition and multiplication are performed modulo the irreducible polynomial $i(t)$.

The three secret parameters in HFE (i.e., its private key) are two affine transformations $S, T : \mathbb{F}^n \to \mathbb{F}^n$ and one private polynomial $P : \mathbb{E} \to \mathbb{E}$, hence the private key K is the triple $(S, P, T) \in \mathrm{AGL}_n(\mathbb{F}) \times \mathbb{E}[x] \times \mathrm{AGL}_n(\mathbb{F})$ where $\mathrm{AGL}_n(\mathbb{F})$ denotes the set of affine transformations over the finite field $\mathbb{F}$. Note: an affine transformation over a finite field is always invertible. The public key k are polynomials $(p_1, \ldots, p_n)$ over $\mathbb{F}$ where each of them consists of n variables $(x_1, \ldots, x_n)$. We describe in Section 3 how the public key can be derived from the private key.

In contrast to the public key, the private polynomial P is defined over the extension field $\mathbb{E}$ and depends on the single variable x. It has degree $d := \partial P$ and some restrictions on its powers:

$$P(x) := \sum_{\substack{0 \le i,j \le d \\ q^i + q^j \le d}} c_{i,j} x^{q^i + q^j} + \sum_{\substack{0 \le k \le d \\ q^k \le d}} b_k x^{q^k} + a$$

$$\text{where} \begin{cases} c_{i,j} x^{q^i+q^j} & \text{for } c_{i,j} \in \mathbb{E} \text{ are the quadratic terms,} \\ b_k x^{q^k} & \text{for } b_k \in \mathbb{E} \text{ are the linear terms, and} \\ a & \text{for } a \in \mathbb{E} \text{ is the constant term} \end{cases}$$

Fig. 1: Structure of the Private Polynomial P in HFE

The purpose of this restriction on the powers is to keep the public polynomials $(p_1, \ldots, p_n)$ "small", i.e., at most quadratic in $(x_1, \ldots, x_n)$. Being quadratic is enough for a system of multivariate equations over a finite fields to be $\mathcal{NP}$-complete [GJ79, p. 251]. On the other hand, as $x^q \to x$ is a linear transformation in $\mathbb{F} = \mathrm{GF}(q)$, the operation $x^{q^i+q^j}$ can be expressed in terms of two linear transformations, i.e., is at most quadratic over $\mathbb{F}$.

As we see in Figure 2, the public key $k = (p_1, \ldots, p_n)$ is used to bypass the private key $K = (S, P, T)$. The point is that the two affine transformations S and T as well as the private

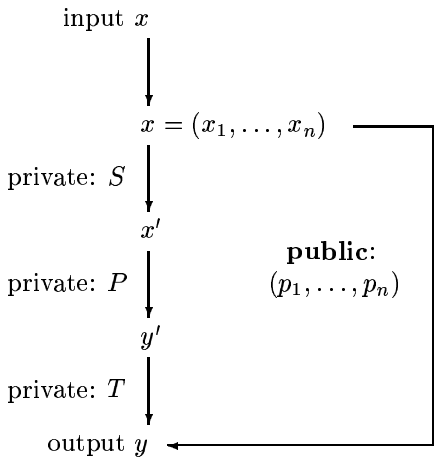

Fig. 2: Private Key (S, P, T) as MQ trapdoor in HFE

polynomial P can be inverted while there are no efficient algorithms available for inverting the public polynomials without knowing the private key [Pat96b, Cou01].

To apply the public key or the private key to a message, this message must be in the correct form. For applying the public key or an affine transformation, it must be represented as a vector $v \in \mathbb{F}^n$ while for applying the private polynomial P, it must be an element $e \in \mathbb{E}$. So we have to transfer elements between $\mathbb{E}$ and $\mathbb{F}^n$. We use a correspondence between corresponding coefficients, i.e., $e_i = f_i$ for e_i the coefficients of an element $e \in \mathbb{E}$ and f_i the coefficients of a vector $f \in \mathbb{F}^n$.

3 Key Generation

After explaining the overall structure of HFE in the previous section, we move on to private key generation. We start with the technique from Matsumoto-Imai [MI88].

3.1 Polynomial Interpolation

We begin with a description of polynomial interpolation for fields $\mathbb{F} \neq \mathrm{GF}(2)$. Key generation for $\mathbb{F} = \mathrm{GF}(2)$ is slightly different, we deal with this case later in this section.

For HFE, we want to obtain polynomials over $\mathbb{F}$ as the public key which have the form

$$p_i(x_1, \ldots, x_n) = \gamma_{i,j,k} x_j x_k + \beta_{i,j} x_j + \alpha_i \,,$$

for $1 \leq i, j, k \leq n$ and $\alpha_i, \beta_{i,j}, \gamma_{i,j,k} \in \mathbb{F}$. To compute these polynomials p_i, we use polynomial interpolation, i.e., we need the output of these polynomials for several inputs. To do so, we exploit that the private key $K = (S, P, T)$ yields the same values as the public key. Therefore, we evaluate the function $T(P(S(x)))$ for several values of x:

- $\eta_0 \in \mathbb{F}^n$ is the 0 vector
- $\eta_j \in \mathbb{F}^n : 1 \leq j \leq n$, is a vector with its j^{th} coefficient 1, the others 0
- $\eta_{j,k} \in \mathbb{F}^n : 1 \leq j < k \leq n$, is a vector with its j^{th} and k^{th} coefficient 1 , the others 0

These $1+n+\frac{1}{2}n(n-1) = \frac{1}{2}n(n+1)+1$ vectors yield the required coefficients, as we see below:

$$\begin{aligned}
T(P(S(\eta_0)))_i &= \alpha_i \\
T(P(S(\eta_j)))_i &= \alpha_i + \beta_{i,j} + \gamma_{i,j,j} \\
T(P(S(a\eta_j)))_i &= \alpha_i + a\beta_{i,j} + a^2\gamma_{i,j,j} \text{ where } a \in \mathbb{F}, a \neq 0,1 \\
T(P(S(\eta_{j,k})))_i &= \alpha_i + \beta_{i,j} + \gamma_{i,j,j} + \gamma_{i,j,k}
\end{aligned}$$

The values for $\alpha_i, \beta_{i,j}, \gamma_{i,j,k}$ are obtained by

$$\begin{aligned}
\alpha_i &:= T(P(S(\eta_0)))_i \\
\gamma_{i,j,j} &:= \frac{1}{a(a-1)}(T(P(S(a\eta_j)))_i - aT(P(S(\eta_j)))_i + (1-a)\alpha_i \\
\beta_{i,j} &:= (T(P(S(\eta_j)))_i - \gamma_{i,j,j} - \alpha_i \\
\gamma_{i,j,k} &:= (T(P(S(\eta_{j,k})))_i - \gamma_{i,j,j} - \gamma_{i,k,k} - \beta_{i,j} - \beta_{i,k} + \alpha_i
\end{aligned}$$

This yields the private polynomials for $\mathbb{F} \neq \mathrm{GF}(2)$.

Now we investigate the overall complexity of this algorithm. For this purpose, keep in mind that we have to evaluate $O(n^2)$ vectors from $\mathbb{F}^n$. Each affine transformation S and T requires $O(n^2)$ steps, which yields $O(n^4)$ operations for the first step, namely, evaluating S. The second step, that is, evaluating P, requires each time $O(n^2)$ multiplications — due to the special choice of the polynomial P. In addition, $O(\log d)$ squaring operations are necessary. As each multiplication requires $O(n^2)$ operations [LD00], and we also evaluate $O(n^2)$ values each, the overall complexity of key generation with polynomial interpolation is $O(n^6)$, as $O(\log d)$ is negligible in comparison to $O(n^2)$.

To adapt the algorithm to $\mathbb{F} = \mathrm{GF}(2)$, we observe that $x^2 = x$ over $\mathrm{GF}(2)$, i.e., all squares in only one variable become linear factors instead. Therefore, we can skip all terms with $\gamma_{i,j,j}$, i.e., all quadratic terms in x_j^2 for $1 \leq j \leq n$. Moreover, we do not have to evaluate $T(P(S(a\eta_j)))_i$ for $a \neq 0,1$ as we do not have to distinguish between quadratic and linear terms in x_i. However, the overall complexity of the algorithm remains the same, namely $O(n^6)$.

3.2 Base Transformation

As pointed out in the previous section, it requires $O(n^6)$ steps to obtain the public key from a given private key by using polynomial interpolation. In this section, we describe a new algorithm for this problem, called "base transformation". The key idea (see the Appendix for a toy example) is to transfer a base of the message space $\mathbb{F}^n$ rather than evaluating $O(n^2)$ vectors. Using base transformation, we obtain the same complexity as for polynomial interpolation. However, the first step, that is, applying affine transformation S, will require only $O(n^2)$ steps. Moreover, our simulations (see Section 4) show that polynomial transformation needs approximately 30% more time.

As already pointed out, we do not apply $\mathrm{HFE}(x)$ to elements from $\mathbb{F}^n$, but to an arbitrary base $\mathfrak{B}$ of $\mathbb{F}^n$. In polynomial notation, the base chosen is

$$\mathfrak{B} = \left\{ \begin{array}{lcccc} p_1 & = & x_1 & & \\ & \vdots & & \ddots & \\ p_n & = & & & x_n \end{array} \right\}$$

and consists of n polynomials over $\mathbb{F}$. Furthermore, the base transformation technique identifies this base with an element of $\mathbb{E} = \mathbb{F}[t]/i(t)$. Here $i(t)$ denotes the irreducible polynomial which

generates $\mathbb{E}$. So the set $\mathfrak{B}$ is also viewed as polynomial

$$\mathfrak{P}(x_1, \ldots, x_n)[t] = p_n t^{n-1} + \ldots + p_1 = x_n t^{n-1} + \ldots + x_1 \, .$$

Another way of thinking about the polynomial $\mathfrak{P}$ is to replace each coefficient a_i (where $a_i \in \mathbb{F}$) by the corresponding polynomial p_i. In this context, it is important to notice the difference between t on the one hand and $x_1, \ldots, x_n$ on the other hand. The first generates the extension field $\mathbb{E}$, while the second is a base of the message space $\mathbb{F}^n$. All operations in the function $\mathrm{HFE}(x) = T(P(S(x)))$ are applied in the same way as they would be applied to elements from $\mathbb{E}$. This means especially that multiplication and squaring are done modulo the irreducible polynomial $i(t)$. The following sections describe this in detail and also deal with the complexity of the operations involved.

3.2.1 Affine Transformation S

When applying S to $\mathfrak{P}$, each coefficient in $\mathfrak{P}$ is multiplied with n elements from the corresponding matrix, hence we would expect a total of $O(n^3)$ operations. By virtue of the choice of $\mathfrak{P}$, this drops to $O(n^2)$, as each polynomial $p_1, \ldots, p_n$ has only one non-zero coefficient. Exploiting the choice of $\mathfrak{P}$ further, transferring S to arbitrary polynomials $p_1, \ldots, p_n$ can be seen as a simple copy operation and hence requires $n^2 + n = O(n^2)$ operations in total.

3.2.2 Applying Polynomial P

After expressing affine transformation S in terms of affine polynomials $p_1, \ldots, p_n$, *i.e.,* in terms of $\mathfrak{P}$, we have to investigate how raising $\mathfrak{P}$ to the power of $q := |\mathbb{F}|$ affects the underlying polynomials. Therefore, we concentrate on one polynomial $p = \alpha + \beta_1 x_1 + \ldots \beta_n x_n$ with $\alpha, \beta_i \in \mathbb{F}$. Using $x^q = x$ and $(a+b)^q = a^q + b^q$ (see [LN00]), we obtain

$$\begin{aligned} p^q &= (\alpha + \beta_1 x_1 + \ldots + \beta_n x_n)^q \\ &= \alpha^q + \beta_1^q x_1^q + \ldots + \beta_n^q x_n^q \\ &= \alpha + \beta_1 x_1 + \ldots + \beta_n x_n \end{aligned}$$

so $p^q = p$ for all polynomials over $\mathbb{F}$. However, $\mathfrak{P}$ also depends on t, and $t^q \neq t$ in general. So $\mathfrak{P}^q$ will yield an arbitrary $\mathfrak{Q}$, which consists of only linear polynomials in $x_1, \ldots, x_n$. Computing $x^{q^i + q^j}$ however, is by no means a linear operation but yields quadratic polynomials in $x_1, \ldots, x_n$. Finally, we have to apply the coefficients, i.e., elements from $\mathbb{E}$ to the result. This requires a further reduction by t but does not change the degree of the polynomials involved.

In terms of complexity, squaring of a polynomial $\mathfrak{P}(x_1, \ldots, x_n)[t]$ requires $O(n^3)$ steps. Multiplying two affine polynomials $\mathfrak{P}$, $\mathfrak{Q}$ is more expensive: it requires $O(n^2)$ multiplications over $\mathbb{E}$ — both for interpolating the result $\mathfrak{R}$ or for computing it directly — and each multiplications requires $O(n^2)$ steps in terms of $\mathbb{F}$. As the private polynomial P has up to $O(n^2)$ many coefficients, the overall complexity is $O(n^6)$. Adding the results $\mathfrak{R}$ from these computations will yield only a complexity of $O(n^4)$ and is hence negligible for our purpose.

3.2.3 Transformation T

As for affine transformation S, we have to apply affine transformation T to the result of Section 3.2.2. This requires $O(n^4)$ operations, as we have to multiply $O(n^2)$ many coefficients with an $(n \times n)$-matrix.

3.2.4 Overall Algorithm

Using the different steps outlined above, we obtain the following algorithm:

1. Express S in terms of affine polynomials, that is, as $\mathfrak{P}$.
2. Compute $\mathfrak{P}^{q^i}$ for $i = 0, 1, \ldots, \lfloor \log_q d \rfloor$.
3. Compute $\mathfrak{P}^{q^k+q^l} = \mathfrak{P}^{q^k}\mathfrak{P}^{q^l}$ using the results of Step 2.
4. Compute $b_i\mathfrak{P}^{q^i}$ and $c_{k,l}\mathfrak{P}^{q^k+q^l}$ using the results from steps 2 and 3.
5. Form the sum of the results of Step 4 and add a.
6. Apply T to the result of Step 5.

Here $a, b_i, c_{j,k} \in \mathbb{E}$ are the coefficients of the private polynomial P (see Section 2).

3.3 Adapted Evaluation as Intermediate Technique

Sections 3.1 and 3.2 show that one can compute the public key for a given private key in $O(n^6)$ steps. The advantage of the base transformation technique is that the number of steps is reduced from $O(n^4)$ to $O(n^2)$ when applying the affine transformation S.

However, it is possible to obtain a similar result without using the base transformation technique. For this we observe that any vector of Hamming weight two is the sum of two vectors of Hamming weight one. Denote $\eta_i \in \mathbb{F}^n$ the vector which has only zeros but one at position i and $\eta_{i,j} \in \mathbb{F}^n$ the vector which has only zeros but one at positions i, j where $1 \leq i \leq n$ and $1 \leq i < j \leq n$, respectively. Moreover, denote η_0 the vector which has zeros only. Using this, we define further:

$$s_0 := S(\eta_0) \quad s_i := S(\eta_i) \quad s_{i,j} := S(\eta_{i,j}) \, .$$

It is easy to see that $s_{i,j} = s_i + s_j - s_0$. In addition, applying S to η_0, η_i does not require any matrix multiplication as s_i can be seen as selecting one column vector rather than a complete matrix multiplication. So obtaining s_0, s_i requires $O(n^2)$ steps, while computing $s_{i,j}$ requires $O(n^2)$ additions, and hence has complexity $O(n^3)$, as each addition requires $O(n)$ steps [LD00].

Remark: In addition, we could exploit the fact that $(a+b)^{q^k} = a^{q^k} + b^{q^k}$ in finite fields (see [LN00]). So it is sufficient to raise s_0, s_i to the power $q^1, \ldots, q^{\lfloor \log_q d \rfloor}$ and then obtain $s_{i,j}^{q^k}$ by adding the corresponding elements. Again, we obtain complexity $O(n^3)$, as we have to perform $O(n^2)$ additions. However, as we do not have to raise the $s_{i,j}$ to the powers of q, the overall number of operations is lower. Nevertheless, if we were doing so, we would need as much code as for the base transformation technique and spoil the overall idea of adapting just a little bit of code. In fact, exploiting this relation, too, would lead to "vector transformation" instead of "base transformation" and we therefore expect a similar complexity and running time for both techniques.

After this remark, we go back to the adapted evaluation technique. In terms of complexity, the algorithm is less efficient than the base transformation technique. However, for the base transformation technique, we need much more specialised code. This is not the case for adapted evaluation which needs as much code as polynomial interpolation. So in terms of the implementation effort versus performance trade-off, the adapted evaluation technique seems to be the best choice of the three options discussed in this section.

4 Speed Comparison

We have implemented all three techniques in Java. Our implementation consists of approx. 9000 lines of Java code, including comments and testing routines [Wol03]. All speed measurements were performed on a Pentium IV-2.6GHz with Java 1.3. The results given in the table below are the average of 101 measurements each.

	GF(2^{103})	GF(2^{131})	GF(2^{151})
Polynomial Interpolation [ms]	6000	13,463	20,585
Base Transformation [ms]	4555	10,303	15,705
Adapted Evaluation [ms]	5921	13,438	20,541

Tab. 1: Timing Results for Different Extension Fields

As we see from the table above, polynomial interpolation has always a 30% higher running time than base transformation. Adapted evaluation is slightly better than normal polynomial interpolation — but only by a small fraction. However, as it does not need any additional memory or more code than polynomial interpolation, it is certainly worth being considered. But as the difference is very small, we expect the underlying architecture to play an important role here.

However, it is interesting to investigate the rather large difference between base transformation and polynomial interpolation in more depth. As both have an overall complexity of $O(n^6)$, we would have expected a far more similar running time for both algorithms. We assume that the reason for this result lies in the fact that the code for base transformation can be optimised further than the code for polynomial interpolation. While the latter consists of very simple, easy to optimise steps, the first consists of many different steps, which can be optimised using various techniques, for example, using an interpolation-like technique for multiplication of affine polynomials. In total, this leads to a lower running time for base transformation.

Unfortunately, we cannot compare our results to other implementations, as [MI88, Pat96b, CGP02a, CGP02b, CGP00] do not provide timing results for key generation. Moreover, the author is not aware of other literature which provides timing for HFE key generation.

5 Adaption to other Systems

In this section, we sketch how base transformation can be adapted to other systems, e.g., the Dragon scheme [Pat96a] or to different versions of HFE [Pat96b].

We always use the same idea: replace the coefficients in the vector space $\mathbb{F}^n$ and the extension field $\mathbb{E}$ by arbitrary polynomials (e.g., by $\mathfrak{B}$ from Section 3.2). After that, apply each step from message encryption with the private key to these polynomials from $\mathfrak{B}$. For example, in HFE-, we proceed as in Section 3.2. As soon as we discard some of the message/signature bits (therefore the name, HFE-), we discard the corresponding polynomials.

HFEv is more difficult, as we have a different private polynomial P namely

$$P_{(z_1,\ldots,z_v)} \quad := \sum_{\substack{0\le i,j\le d\\ q^i+q^j\le d}} \alpha_{i,j} x^{q^i+q^j} + \sum_{\substack{0\le k\le d\\ q^k\le d}} \beta_k(z_1,\ldots,z_v) x^{q^k} + \gamma(z_1,\ldots,z_v)\,,$$

for $\alpha_{i,j} \in \mathbb{E}$, $\beta_k(z_1,\ldots,z_v)$ are affine in $(z_1,\ldots,z_v)$, and $\gamma(z_1,\ldots,z_v)$ is at most quadratic in $(z_1,\ldots,z_v)$. Here $(z_1,\ldots,z_v)$ are called "vinegar" variables. These vinegar variables are over $\mathbb{F}$. Moreover, $S \in \mathrm{AGL}_{n+v}(\mathbb{F})$ but $T \in \mathrm{AGL}_n(\mathbb{F})$.

However, if we proceed as described above, we apply the affine transformation S to $\mathfrak{B}$ (now with $n+v$ polynomials), and then set $z_1 := p_{n+1}, \ldots, z_v := p_{n+v}$. After that, we compute the public key as described in Section 3.2. However, in Step 4 of Section 3.2.4, we have to multiply vectors of affine polynomials. This is similar to Step 3 in the same algorithm.

The technique "base transformation" is quite general, i.e., it is not only applicable to variations of HFE.

We conclude this overview with the Dragon scheme [Pat96a]. In contrast the HFE, the public key has the form

$$\sum \alpha_{i,j,k,l} x_j x_k y_l + \sum \beta_{i,j,k} x_j x_k + \sum \gamma_{i,j,k} x_j y_k + \sum \delta_{i,j} x_j + \sum \epsilon_{i,j} y_j + \zeta_i = 0\,,$$

for $1 \leq i, j, k, l \leq n$ and $\alpha_{i,j,k,l}, \beta_{i,j,k}, \gamma_{i,j,k}, \delta_{i,j}, \epsilon_{i,j}, \zeta_i \in \mathbb{F}$. Here, the parameter i identifies the equation, i.e, the public key consists of n equations of the above type. The private key consists of two affine transformations $S, T \in \mathrm{AGL}_n(\mathbb{F})$ and an equation $E : x^{q^a+q^b} y = x^{q^c+q^d} y$ with $a, b, c, d \in \mathbb{N}$.

To apply the base transformation, we apply the affine transformations S and T^{-1} to a polynomial base of $\mathbb{F}^n$. However, we have to distinguish between $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ for this purpose. After that, we apply the equation E, *i.e.,* we calculate the public key in terms of multiplication of affine polynomials (see Section 3.2.2). A sketch of this algorithm can also be found in [Pat96a].

6 Conclusions

In this paper, we presented two novel techniques called "base transformation" and "adapted evaluation". Both are faster than the previously known "polynomial interpolation" technique. Table 2 summarises our results, setting the running time of base transformation equal to 1.

	Theoretical Estimation	Running Time
Polynomial Interpolation	$\mathrm{O}(n^4) + \mathrm{O}(n^6) + \mathrm{O}(n^4)$	≈ 1.3
Base Transformation	$\mathrm{O}(n^2) + \mathrm{O}(n^6) + \mathrm{O}(n^4)$	1.0
Adapted Evaluation	$\mathrm{O}(n^3) + \mathrm{O}(n^6) + \mathrm{O}(n^4)$	≈ 1.3

Tab. 2: Summary of the Results for Key Generation Algorithms

The base transformation technique is the fastest from our test set — both from a theoretical perspective and also in terms of the time measurements, *cf* Section 4. The other two algorithms have a roughly 30% higher running time. There is a slight difference between polynomial interpolation and adapted evaluation. However, it is smaller than 1% (see Section 4).

Moreover, the base transformation is quite general. As outlined in Section 5, it can be adapted to various other systems which also use multivariate quadratic equations as their public key. Therefore, it is useful for run-time efficient public key generation in such systems. However, for code-efficient implementations, polynomial interpolation and adapted evaluation are better choices.

Acknowledgements

We want to thank Simon Foley and Patrick Fitzpatrick (University College Cork, Ireland) for fruitful discussions and helpful remarks. Moreover, we want to thank Bart Preneel and Michael Quisquater (K.U. Leuven) for helpful remarks.

The author is funded by a research grant of the K.U.Leuven.

References

[CDF03] Nicolas T. Courtois, Magnus Daum, and Patrick Felke. On the security of HFE, HFEv- and Quartz. In *Public Key Cryptography - PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 337–350. Y. Desmedt, editor, Springer, 2002. `http://eprint.iacr.org/2002/138`.

[CGP00] Nicolas Courtois, Louis Goubin, and Jacques Patarin. *Flash: Primitive specification and supporting documentation*, 2000. `https://www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions/flash.zip`, 9 pages.

[CGP02a] Nicolas Courtois, Louis Goubin, and Jacques Patarin. *Quartz: Primitive specification (second revised version)*, 2002. `https://www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions/quartzv21-b.zip`, 17 pages.

[CGP02b] Nicolas Courtois, Louis Goubin, and Jacques Patarin. *SFlash: Primitive specification (second revised version)*, 2002. `https://www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions/sflashv2.zip`, 11 pages.

[Cou01] Nicolas T. Courtois. The security of Hidden Field Equations (HFE). In *The Cryptographer's Track at RSA Conference 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 266–281. D. Naccache, editor, Springer, 2001. `http://www.minrank.org/hfesec.{ps|dvi|pdf}`.

[Fau02] Jean-Charles Faugère. HFE challenge 1 broken in 96 hours. Announcement that appeared in `news://sci.crypt`, 19$^{\text{th}}$ of April 2002.

[GJ79] Michael R. Garay and David S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979. ISBN 0-7167-1044-7 or 0-7167-1045-5.

[LD00] Julio Lopéz and Ricardo Dahab. An overview of elliptic curve cryptography. Technical report, Institute of Computing, State University of Campinas, Brazil, 22$^{\text{nd}}$ of May 2000. `http://citeseer.nj.nec.com/333066.html` or `http://www.dcc.unicamp.br/ic-tr-ftp/2000/00-14.ps.gz`.

[LN00] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, 2000. ISBN 0-521-46094-8.

[MI88] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynomial-tuples for efficient signature verification and message-encryption. In *Advances in Cryptology – EUROCRYPT 1988*, volume 330 of *Lecture Notes in Computer Science*, pages 419–545. Christoph G. Günther, editor, Springer, 1988.

[Pat95] Jacques Patarin. Cryptanalysis of the Matsumoto and Imai public key scheme of Eurocrypt'88. In *Advances in Cryptology – CRYPTO 1995*, volume 963 of *Lecture Notes in Computer Science*, pages 248–261. Don Coppersmith, editor, Springer, 1995.

[Pat96a] Jacques Patarin. Asymmetric cryptography with a hidden monomial. In *Advances in Cryptology – CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 45–60. Neal Koblitz, editor, Springer, 1996.

[Pat96b] Jacques Patarin. Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms. In *Advances in Cryptology – EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Ueli Maurer, editor, Springer, 1996. Extended Version: `http://www.minrank.org/hfe.pdf`.

[Wol03] Christopher Wolf. Java implementation of key generation for HFE, source code. `http://www.christopher-wolf.de/hfe_kg/`, ≈9000 lines of code, 2003.

Appendix

Example of Public Key Generation

This section gives a toy example of public key generation, using the algorithm from Section 3.2. The parameters are: $\mathbb{F}$ =GF(2), $i(t) := t^3 + t + 1$, $\mathbb{E} = \mathbb{F}[t]/i(t)$, and $P(x) = x^3 + (t+1)x + 1$. Here $(t+1) \in \mathbb{E}$; as as bit string, it would be denoted $[011]_b$. Moreover, let

$$\begin{aligned} S(x_1, x_2, x_3) &:= \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \\ T(x_1, x_2, x_3) &:= \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \end{aligned}$$

By applying the algorithm from Section 3.2, we obtain

$$\mathfrak{P}(x_1, x_2, x_3) = \begin{pmatrix} p_1 := x_1 + x_3 \\ p_2 := x_2 + 1 \\ p_3 := x_2 + x_3 \end{pmatrix}$$

as a representation of S in terms of three polynomials p_1, p_2, p_3. This can equally be seen as

$$\mathfrak{P}[t] = (x_2 + x_3)t^2 + (x_2 + 1)t + (x_1 + x_3)$$

to stress the point that we want to mimic operations in $\mathbb{E}$. First we compute $\mathfrak{P}[t]^2$:

$$\begin{aligned} \mathfrak{P}[t]^2 &= [(x_2 + x_3)t^2 + (x_2 + 1)t + (x_1 + x_3)]^2 \\ &= (x_2 + x_3)t^4 + (x_2 + 1)t^2 + (x_1 + x_3) \\ &\equiv (x_3 + 1)t^2 + (x_2 + x_3)t + (x_1 + x_3), \end{aligned}$$

where $\equiv$ denotes reduction by $i(t) := t^3 + t + 1$. Furthermore,

$$\begin{aligned} \mathfrak{P}[t]^3 &= \mathfrak{P}[t]^2 \mathfrak{P}[t] \\ &= [(x_3 + 1)t^2 + (x_2 + x_3)t + (x_1 + x_3)][(x_2 + x_3)t^2 + (x_2 + 1)t + (x_1 + x_3)] \\ &= (x_2 + x_2x_3)t^4 + (x_2x_3 + 1)t^3 + (x_1 + x_1x_2)t^2 + (x_1 + x_1x_3)t + (x_1 + x_3) \\ &\equiv (x_1 + x_2 + x_1x_2 + x_2x_3)t^2 + (x_1 + x_2 + x_1x_3 + 1)t + (x_1 + x_3 + x_2x_3 + 1). \end{aligned}$$

As x in $P(x)$ has a constant multiple $(t+1)$, we have to reflect this in terms of $\mathfrak{P}[t]$:

$$\begin{aligned} (t+1)\mathfrak{P}[t] &= (t+1)[(x_2 + x_3)t^2 + (x_2 + 1)t + (x_1 + x_3)] \\ &= (x_2 + x_3)t^3 + (x_3 + 1)t^2 + (x_1 + x_2 + x_3 + 1)t + (x_1 + x_3) \\ &\equiv (x_3 + 1)t^2 + (x_1 + 1)t + (x_1 + x_2). \end{aligned}$$

So as an overall result, we can compute $P(x)$ in terms of $\mathfrak{P}[t]$ and denote the result with $\mathfrak{Q}[t]$:

$$\begin{aligned} \mathfrak{Q}[t] &:= \mathfrak{P}[t]^3 + (t+1)\mathfrak{P}[t] + 1 \\ &= [(x_1 + x_2 + x_1x_2 + x_2x_3)t^2 + (x_1 + x_2 + x_1x_3 + 1)t + (x_1 + x_3 + x_2x_3 + 1)] + \\ &\quad [(x_3 + 1)t^2 + (x_1 + 1)t + (x_1 + x_2)] + 1 \\ &= (x_1 + x_2 + x_3 + x_1x_2 + x_2x_3 + 1)t^2 + (x_2 + x_1x_3)t + (x_2 + x_3 + x_2x_3). \end{aligned}$$

Before we can apply affine transformation T, we have to change our point of view and we have to think about $\mathfrak{Q}[t]$ as a vector with 3 rows, denoted $\mathfrak{Q}(x_1, x_2, x_3)$:

$$\mathfrak{Q}(x_1, x_2, x_3) := \begin{pmatrix} q_1 & := & x_2 + x_3 + x_2x_3 \\ q_2 & := & x_2 + x_1x_3 \\ q_3 & := & x_1 + x_2 + x_3 + x_1x_2 + x_2x_3 + 1 \end{pmatrix}.$$

The affine transformation T can now be applied by ordinary matrix multiplication and vector addition. This step yields result $\mathfrak{R}$:

$$\begin{aligned} \mathfrak{R}(x_1, x_2, x_3) &:= T(\mathfrak{Q}(x_1, x_2, x_3)) \\ &= \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} q_2 + q_3 + 1 \\ q_1 + q_2 \\ q_3 \end{pmatrix} \\ &= \begin{pmatrix} x_1 + x_3 + x_1x_2 + x_1x_3 + x_2x_3 \\ x_3 + x_1x_3 + x_2x_3 \\ x_1 + x_2 + x_3 + x_1x_2 + x_2x_3 + 1 \end{pmatrix}. \end{aligned}$$

As we see, each row in the vector $\mathfrak{R}$ consists of one polynomial with at most quadratic terms in x_1, x_2, x_3. By construction, $\mathfrak{R}(x_1, x_2, x_3) = T(P(S(x_1, x_2, x_3)))$ for any $(x_1, x_2, x_3) \in \mathbb{F}^3$.