Efficient Public Key Generation for Multivariate Cryptosystems

Christopher Wolf[1]
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/`

Cryptology ePrint Archive, Report 2003/089
`http://eprint.iacr.org/`

Current Version: 2005-08-06
First Version: 2003-05-05

This is a preliminary version of the article [Wol04]
"Efficient public key generation for HFE and variations."
In Cryptographic Algorithms and Their Uses 2004, *pages 78–93.*
Dawson, Klimm, editors, QUT University, 2004.

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 with 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 "polynomial interpolation" with our new developments by empirical studies. We conclude that the running time of polynomial interpolation is approximately 30% higher than for base transformation.

Keywords: Multivariate Cryptography, Hidden Field Equations, Key Generation

[1]This project has mainly been carried out while the author was at University College Cork, Ireland

1 Introduction

During the past years, several asymmetric cryptographic systems were proposed based on multivariate cryptography [MI88, Pat96a, Pat96b, CGP01, KPG99, Moh99, CGP03, YC04, KS04]. 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 "$\mathcal{MQ}$-problem", *i.e.*, multivariate quadratic equations over finite fields are difficult to solve [GJ79]. Although some of these systems have been successfully broken (*e.g.*, [Pat95, KS99, GC00, FJ03, 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 [CGP01]). 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 has not yet received much attention. For example, [Pat96b, CGP01, CGP03] completely ignore it. However, [MI88] presents a quite general algorithm which can be adapted easily to different systems. In a nutshell, their algorithm uses polynomial interpolation for multivariate polynomials. See Sect. 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 a technique sketched in [Pat96a]. We demonstrate "base transformation" for the "Hidden Field Equations" (HFE) system [Pat96b] and outline how it can be adapted to variations of it. 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 by simulations.

This paper is organised as follows: in Sect. 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. Sect. 4 presents a speed comparison using a Java implementation of all three techniques. In Sect. 5 we sketch how the base transformation technique can be adapted to variations of HFE. The paper concludes with Sect. 6.

2 Description of the HFE system

This section gives a concise description of the "Hidden Field Equations" system; a more detailed description can be found in [Pat96b]. 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 the extension field $\mathbb{E}$, addition is normal polynomial addition and multiplication is 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 in n variables over the finite field $\mathbb{F}$. *Note:* an affine transformation over a finite field is always invertible. The public key k is an n-tuple of polynomials $(p_1, \ldots, p_n)$ over $\mathbb{F}$ where each polynomial consists of n variables $(x_1, \ldots, x_n)$. In a nutshell, we want $\forall x \in \mathbb{F}^n : k(x) = T(P(S(x)))$, *i.e.*, the private key and the public key yield the same result if applied to the same element x of the vector space $\mathbb{F}^n$. We describe in Sect. 3 how this public key k can be derived from the private key K.

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:

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

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

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

Our aim in restricting the powers of P 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 field to be $\mathcal{NP}$-complete (*cf* [GJ79, p. 251] and [PG97, App.] for a proof). On the other hand, as $x^q \to x$ is a linear transformation in the finite field $\mathbb{F} = \mathrm{GF}(q)$, 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 already pointed out, the public key is a composition of the private key. In this context, the two affine transformations S, T are expressed as affine systems of multivariate equations in n variables each, denoted $\mathfrak{S}, \mathfrak{T}$. The private polynomial P is expressed as a quadratic system of multivariate equations, denoted $\mathfrak{P}$. The public key is then computed as the composition $\mathfrak{T} \circ \mathfrak{P} \circ \mathfrak{S}$. As both $\mathfrak{S}, \mathfrak{T}$ are of degree 1, while $\mathfrak{P}$ is of degree 2, the overall degree of this composition is of degree 2. This idea is summarised in Figure 2. By changing our point of view we can say

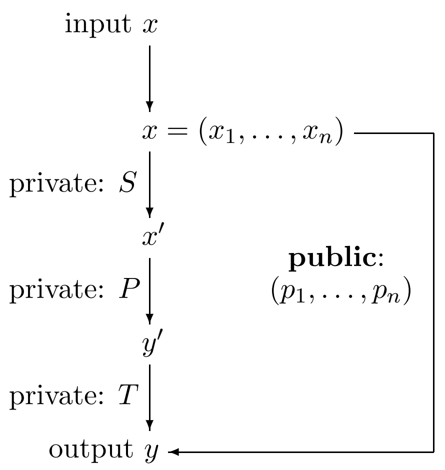

Fig. 2: Private Key (S, P, T) as $\mathcal{MQ}$ trapdoor in HFE

that the public key $k = (p_1, \ldots, p_n)$ is used to "bypass" the private key $K = (S, P, T)$ without revealing its individual components S, P, T. The point is that the two affine transformations S and T as well as the private polynomial P can be inverted while there are no efficient algorithms available for inverting the public polynomials without knowing the private key [Pat96b, Cou01]. A way of obtaining S, P, T from the public key only is presented in [KS99]. However, it is far from being efficient for suitable choices of parameters in HFE [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 $f \in \mathbb{F}^n$ throughout this paper.

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]. The other techniques described in this paper have been developed by the author.

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.

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 some $\alpha_i, \beta_{i,j}, \gamma_{i,j,k} \in \mathbb{F}$. 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 $1 + n + \frac{1}{2}n(n+1) = \frac{1}{2}(n+1)(n+2)$ vectors yield the required coefficients, as we see below:

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

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

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

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

To adapt the algorithm to $\mathbb{F} = \mathrm{GF}(2)$, we observe that $x^2 = x$ over GF(2), *i.e.*, all squares in only one variable become linear factors. 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.

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 using polynomial interpolation. In this section, we describe a new algorithm for this problem, called "base transformation". The key idea is to transfer a base of the message space $\mathbb{F}^n$ rather than evaluating $O(n^2)$ vectors. This technique has been developed by the author.

As already pointed out, we do not apply 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

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

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 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 by n elements from the corresponding matrix. By virtue of the choice of $\mathfrak{P}$, each polynomial $p_1, \ldots, p_n$ has only one $p_1, \ldots, p_n$ can be seen as a simple copy operation.

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 $(a+b)^q = a^q + b^q$ (cf [LN00]), we obtain

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

so $p^q = p$ for all polynomials over $\mathbb{F}$. However, the vector $\mathfrak{P}$ also depends on t, and $t^q \neq t$ in general. So the operation $\mathfrak{P}^q$ will yield an arbitrary vector $\mathfrak{Q}$, 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.

3.2.3 Transformation T

As for affine transformation S, we have to apply affine transformation T to the result of Sect. 3.2.2.

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 Sect. 2).

3.3 Adapted Evaluation as Intermediate Technique

Sect. 3.1 and 3.2 show how to compute the public key for a given private key. The advantage of the base transformation technique is that it is faster (see below). However, it needs a lot of specialised code, as we saw in the previous section.

In this section, we outline a third possibility which does not need so much specialised code. It has also been developed by the author. For this technique 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:

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

We 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.

Remark: In addition, we could exploit the fact that $(a+b)^{q^k} = a^{q^k} + b^{q^k}$ in finite fields (see [LN00]). Hence, 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. 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 (see performance results below). However, for the base transformation technique, we need much more specialised code: we need to implement all steps as outlined in the previous section. In our implementation, we needed about about 30 times more code — and therefore programming time — for base transformation in comparison to polynomial interpolation. In addition, the base transformation technique has to be adapted specifically to the system in question while polynomial interpolation may be used quite generic: as long as there is a way of *encrypting* data with the private key alone, polynomial interpolation can use this as a "black box" to derive the corresponding public key. This encryption function is usually quite short — typically 15–30 lines of code — assuming that the building blocks (*e.g.*, affine transformations, finite field arithmetic) already exist. This is not the case for base transformation: here, we need to write specific code to "emulate" finite field arithmetic for multivariate polynomials. So in terms of the implementation effort versus performance (both translates into costs: either paying programmers for fast software or hardware to compensate for the slower key generation), the adapted evaluation technique seems to be worthwhile considering. But as we see below, the difference between adapted evaluation and polynomial interpolation is rather small.

	GF(2^{103})	GF(2^{131})	GF(2^{151})
Polynomial Interpolation [ms]	6001	13,463	20,585
Base Transformation [ms]	4556	10,304	15,706
Adapted Evaluation [ms]	5922	13,439	20,541

Table 1: Timing results on a Pentium IV-2.6GHz with Java 1.3

	GF(2^{103})	GF(2^{131})	GF(2^{151})
Polynomial Interpolation [ms]	7471	16,622	25,495
Base Transformation [ms]	5706	12,635	19,324
Adapted Evaluation [ms]	7436	16,548	25,372

Table 2: Timing results on an Athlon XP 2000+ with Java 1.3

	GF(2^{103})	GF(2^{131})	GF(2^{151})
Polynomial Interpolation [ms]	5962	13,891	20,959
Base Transformation [ms]	4625	10,711	16,085
Adapted Evaluation [ms]	5951	13,877	20,928

Table 3: Timing results on an Athlon XP1700+ with Java 1.4

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.[2] To see how the underlying architecture effects the running time, we used different machines to compare the performance of our implementation both with Java 1.3 and Java 1.4. In all cases, the private polynomial had degree 129. This value was chosen as it is the recommended parameter in Quartz [CGP01]. The first field size 2^{103} is also due to Quartz, the other two values (2^{131} and 2^{151}, respectively) were chosen to see how the implementation scales up for larger fields. The results given in tables 1–5 and are the average of 101 measurements each.

As we see from these tables, polynomial interpolation has always a 30% higher running time than base transformation — independently from the underlying architecture, *i.e.*, processor and Java-version used. Adapted evaluation is always slightly better than normal polynomial interpolation — but the difference is less than 1%. However, as it does not

[2]In the final version of this paper, there is a link to the code

	GF(2^{103})	GF(2^{131})	GF(2^{151})
Polynomial Interpolation [ms]	4202	10,001	15,476
Base Transformation [ms]	3266	7794	11,863
Adapted Evaluation [ms]	4191	9992	15,258

Table 4: Timing results on a HP PA-RISC-700 with Java 1.4 (32 bit)

	$GF(2^{103})$	$GF(2^{131})$	$GF(2^{151})$
Polynomial Interpolation [ms]	6258	10,045	15,605
Base Transformation [ms]	4845	7843	11,957
Adapted Evaluation [ms]	6247	10,018	15,278

Table 5: Timing results on a HP PA-RISC-700 with Java 1.4 (64 bit)

need any additional memory or more code than polynomial interpolation, it is certainly worth being considered.

However, it is interesting to investigate the rather large difference between base transformation and polynomial interpolation in more depth. In our experiments, we identified two main sources for the higher speed of base-transformation:

Memory: For base transformation, we are able to store the results of the exponentiation (cf Sect. 3.2.4, Step 2). This way, we neither need to evaluate the affine transformation S several times nor compute the power-function more than $\log_q d$ times. Both takes only a fraction of a ms (correct value depending on the architecture), but adds up when evaluated $\binom{n}{2} + n + 1$ times. The additional memory requirement here is $\lfloor \log_q d \rfloor n(n+1)$ as we have to store $\lfloor \log_q d \rfloor$ different n-vectors of affine polynomials in n variables each.

Multiplication: For polynomial interpolation, we used a state of the art implementation of finite field arithmetic, as described in [LD00]. There, multiplication in the polynomial base takes n shift operations and n field additions on average. For base transformation, we exploit the fact that we deal with a whole vector of polynomials at a time. This way, we can use an interpolation-like approach (!) for multiplication which turns out to be twice as fast as naïve multiplication. Therefore, we identify this as another important source of the speed-up.

In addition, we notice that the running time on the Athlon 1700+ is *lower* than on the Athlon 2000+. We credit this to the fact that we used Java 1.3 in the latter case. Moreover, we see that the 64-bit-version of Java needs 50% *more* time for $GF(2^{103})$ than the 32-bit-version. This difference nearly vanishes (less than 1% time difference) for $GF(2^{131})$ and $GF(2^{151})$. At present, we do not have a satisfying explanation for this behaviour. However, we re-run the programme several times on different machines in the HP-cluster and always got the same picture: $GF(2^{103})$ with 64-bit needs about 50% more time than $GF(2^{103})$ with 32-bit. Although we cannot explain this difference at present, we want to point out that it is not important for our overall goal: comparing the speed of the three different techniques for public-key generation outlined in this paper on a fixed architecture.

Unfortunately, we cannot compare our results to other implementations, as [MI88, Pat96b, CGP01, CGP00, CGP03] do not provide timing results for key generation. Moreover, the author is not aware of other literature which provides timing results for HFE key generation.

5 Variations of HFE

In this section, we sketch how base transformation can be adapted 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 Sect. 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 Sect. 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

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

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 [KPG99, Sect. 12]. These vinegar variables are over $\mathbb{F}$. Moreover, $T \in \mathrm{AGL}_n(\mathbb{F})$ but $S \in \mathrm{AGL}_{n+v}(\mathbb{F})$.

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 Sect. 3.2. Now, in Step 4 of Sect. 3.2.4, we have to multiply vectors of affine polynomials. This is similar to Step 3 in the same algorithm.

6 Conclusions

In this paper, we presented two novel techniques called "base transformation" and "adapted evaluation". The first is clearly faster than the previously known "polynomial interpolation" technique. Table 6 summarises our results, setting the running time of base transformation equal to 1. This table is based on results from 5 different architectures (cf Sect. 4). We notice that base transformation is the fastest technique from our test set. 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 Sect. 4) and therefore not reflected in the above table.

Moreover, base transformation is quite general. As outlined in Sect. 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

	Running Time
Polynomial Interpolation	≈ 1.3
Base Transformation	1.0
Adapted Evaluation	≈ 1.3

Table 6: Summary of the Results for Key Generation Algorithms

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 An Braeken, Bart Preneel, Michael Quisquater (K.U. Leuven) and Lynn Batten (University of Deakin, Australia) for helpful remarks.

This work was supported in part by the Concerted Research Action (GOA) Mefisto-2000/06 of the Flemish Government and the German Academic Exchange Service (DAAD).

References

[BWP05] An Braeken, Christopher Wolf, and Bart Preneel. A study of the security of Unbalanced Oil and Vinegar signature schemes. In *The Cryptographer's Track at RSA Conference 2005*, Lecture Notes in Computer Science. Alfred J. Menezes, editor, Springer, 2005. 13 pages, cf `http://eprint.iacr.org/2004/222/`.

[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.

[CGP01] Nicolas Courtois, Louis Goubin, and Jacques Patarin. *Quartz: Primitive specification (second revised version)*, October 2001. `https://www.cosic.esat.kuleuven.ac.be/nessie` Submissions, Quartz, 18 pages.

[CGP03] Nicolas Courtois, Louis Goubin, and Jacques Patarin. *SFlash^{v3}, a fast asymmetric signature scheme — Revised Specificatoin of SFlash, version 3.0*, October 17th 2003. ePrint Report 2003/211, `http://eprint.iacr.org/`, 14 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}`.

[FJ03] Jean-Charles Faugère and Antoine Joux. Algebraic cryptanalysis of Hidden Field Equations (HFE) using gröbner bases. In *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Dan Boneh, editor, Springer, 2003.

[GC00] Louis Goubin and Nicolas T. Courtois. Cryptanalysis of the TTM cryptosystem. In *Advances in Cryptology — ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 44–57. Tatsuaki Okamoto, editor, Springer, 2000.

[GJ79] Michael R. Garey 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.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar signature schemes. In *Advances in Cryptology — EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 206–222. Jacques Stern, editor, Springer, 1999.

[KS99] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem. In *Advances in Cryptology — CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Michael Wiener, editor, Springer, 1999. `http://www.minrank.org/hfesubreg.ps` or `http://citeseer.nj.nec.com/kipnis99cryptanalysis.html`.

[KS04] Masao Kasahara and Ryuichi Sakai. A construction of public key cryptosystem for realizing ciphtertext of size 100 bit and digital signature scheme. *IEICE Trans. Fundamentals*, E87-A(1):102–109, January 2004. Electronic version: `http://search.ieice.org/2004/files/e000a01.htm\#e87-a,1,102`.

[LD00] Julio Lopéz and Ricardo Dahab. An overview of elliptic curve cryptography. Technical report, Institute of Computing, State University of Campinas, Brazil, 22nd 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.

[Moh99] T. Moh. A public key system with signature and master key function. *Communications in Algebra*, 27(5):2207–2222, 1999. Electronic version: `http://citeseer/moh99public.html`.

[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`.

[PG97] Jacques Patarin and Louis Goubin. Trapdoor one-way permutations and multivariate polynomials. In *International Conference on Information Security and Cryptology 1997*, volume 1334 of *Lecture Notes in Computer Science*, pages 356–368. International Communications and Information Security Association, Springer, 1997. Extended Version: `http://citeseer.nj.nec.com/patarin97trapdoor.html`.

[WBP04] Christopher Wolf, An Braeken, and Bart Preneel. Efficient cryptanalysis of RSE(2)PKC and RSSE(2)PKC. In *Conference on Security in Communication Networks — SCN 2004*, Lecture Notes in Computer Science, pages 145–151, September 8–10 2004. Extended version: `http://eprint.iacr.org/2004/237`.

[Wol02] Christopher Wolf. *Hidden Field Equations* (HFE) - variations and attacks. Diplomarbeit, Universität Ulm, December 2002. `http://www.christopher-wolf.de/dpl`, 87 pages.

[Wol04] Christopher Wolf. Efficient public key generation for HFE and variations. In *Cryptographic Algorithms and Their Uses 2004*, pages 78–93. Dawson, Klimm, editors, QUT University, 2004.

[WP04] Christopher Wolf and Bart Preneel. Asymmetric cryptography: Hidden Field Equations. In *European Congress on Computational Methods in Applied Sciences and Engineering 2004*. P. Neittaanmäki, T. Rossi, S. Korotov, E. Oñate, J. Périaux, and D. Knörzer, editors, Jyväskylä University, 2004. 20 pages, extended version: `http://eprint.iacr.org/2004/072/`.

[WP05a] Christopher Wolf and Bart Preneel. Equivalent keys in HFE, C^*, and variations. In *Proceedings of Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 33–49. Serge Vaudenay, editor, Springer, 2005. Extended version `http://eprint.iacr.org/2004/360/`, 12 pages.

[WP05b] Christopher Wolf and Bart Preneel. Superfluous keys in $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic asymmetric systems. In *Public Key Cryptography — PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 275–287. Serge Vaudenay, editor, Springer, 2005. Extended version `http://eprint.iacr.org/2004/361/`.

[WP05c] Christopher Wolf and Bart Preneel. Taxonomy of public key schemes based on the problem of multivariate quadratic equations. Cryptology ePrint Archive, Report 2005/077, 12^{th} of May 2005. `http://eprint.iacr.org/2005/077/`, 64 pages.

[YC04] Bo-Yin Yang and Jiun-Ming Chen. Rank attacks and defence in Tame-like multivariate PKC's. Cryptology ePrint Archive, Report 2004/061, 23^{rd} March 2004. `http://eprint.iacr.org/`, 21 pages.