Diplomarbeit

"Hidden Field Equations" (HFE) - Variations and Attacks

Betreuer:

Prof. Dr. Uwe Schöning

Fakultät für Informatik, Universität Ulm

&

Prof. Dr. Patrick Fitzpatrick

Department of Mathematics, University College Cork

von Christopher Wolf

2. Dezember 2002

Abstract

This thesis (Diplomarbeit) describes a cryptographic system with a public/private key structure. The system is based on polynomials over finite fields and called "Hidden Field Equations" (HFE).

The overall idea of cryptography is described briefly in the first chapter of this thesis. Moreover, this chapter gives a short explanation about the RSA system (Rivest, Shamir, Adleman) and also the ElGamal system. Both can be used for encryption/decryption and signature generation/verification.

The same is true for the HFE system which is explained in the next chapter. This chapter describes how n polynomials $(p_1, \ldots, p_n)$ in n variables over a small finite field $\mathbb{F}$ can be used as a public key. The corresponding problem is called "MQ" (i.e., multivariable quadratic). Recommended values for the public key are $n = 80\ldots129$ and $\mathbb{F} = \mathrm{GF}(2)$. In addition, a trapdoor (private key) for this public key is described. This trapdoor consists of two affine transformations, denoted S and T, and also one private polynomial, denoted P, over an extension field $\mathbb{E}$. Using this trapdoor, both encryption and signature generation can be performed. The second chapter also considers the speed of HFE, its security parameters, and which difficulties arise for non-surjective versions of HFE.

In Chapter 3, the complexity of HFE in terms of $\mathcal{NP}$-completeness is discussed. It contains a proof that MQ problem over finite fields is $\mathcal{NP}$-complete. In addition, attacks against HFE are discussed. These attacks are classified as either structural (i.e., revealing the private key) or inversion (i.e., computing signatures or decrypting messages for eavesdropper) attacks.

To increase the security of HFE, it is possible to vary the system. Some of these possible modifications are discussed in the forth chapter. Unfortunately, some of the variations investigated are proven to be less secure than HFE itself. On the other hand, some of them can reduce the public key size or decrease the time for private key computations, which makes them worthwhile in some application domains.

Chapter 5 shows some real world applications of HFE for signature and outlines how HFE can be used for encryption. These real world applications include the two signature algorithms Quartz and SFlash which are currently under consideration in the European cryptographic project NESSIE.

In its last chapter, this thesis is summarised and some fields of possible further research are discussed.

Contents

1 Introduction **5**
- 1.1 Public Key Cryptography 6
- 1.2 Factorisation 7
- 1.3 Discrete Logarithm 9

2 General Hidden Field Equations **11**
- 2.1 Trapdoor in HFE 11
- 2.2 Encryption and Decryption of Messages using the Private Key 13
- 2.3 Public Key: Generation and Encryption 16
  - 2.3.1 Generation by Polynomial Interpolation 16
  - 2.3.2 Base Transformation 18
  - 2.3.3 Encryption 25
- 2.4 Message Signature 25
- 2.5 Speed Considerations 27
- 2.6 Security Parameters 28
- 2.7 Key Size 29
- 2.8 Key Space 29

3 Attacks on HFE **32**
- 3.1 Types of Attacks 32
- 3.2 $\mathcal{NP}$-Completeness of MQ 33
  - 3.2.1 MQ over GF(2) 33
  - 3.2.2 MQ over Domains 35
  - 3.2.3 Conclusions 37
- 3.3 $\mathcal{NP}$ structural attack 38
- 3.4 Linear Attack 39
- 3.5 Patarin Attacks 40
  - 3.5.1 Attack against Matsumoto-Imai 40
  - 3.5.2 Affine Multiple Attack 41

3.5.3 Quadratic Attack 42
3.6 Relinearization and beyond 44
3.6.1 Relinearization 44
3.6.2 XL and FXL algorithm 45
3.6.3 Gröbner Bases 47
3.7 Kipnis-Shamir Attack 48

4 Variations **52**
4.1 Bijection between $\mathbb{F}^n$ and $\mathbb{E}$ revisited 52
4.2 Working with a Subfield $\widetilde{\mathbb{F}}$ of $\mathbb{F}$ 53
4.3 Modifying the Private Polynomial 53
4.4 HFE with More than One Branch 55
4.5 HFE-: Hiding Public Equations 56
4.6 HFE+: Adding Public Equations 58
4.7 HFEv: Adding Vinegar Variables 60
4.8 HFEf: Fixing Input Variables 61
4.9 HFEm: Imitating Vinegar Variables for Encryption 63
4.10 HFEm', HFEz: Zero Added Equations 65
4.11 Modifications Revisited 67

5 Applications **69**
5.1 Flash / SFlash 69
5.2 Quartz 71
5.3 Shared Key Generation 76
5.4 eHFE: HFE for Encryption 79

6 Conclusions **81**

Chapter 1

Introduction

While cryptography during the last millennia was mainly driven by military applications, its use shifted during the last decades to more commercial and public usages. The two keywords “eCommerce” and “PGP” may be enough to point out this new application domain.

The aim of this thesis (Diplomarbeit) is to provide an overview about the public key encryption system “Hidden Field Equations (HFE)”, to investigate further some attacks on this scheme, and also variations which counter these attacks. HFE was proposed in 1996 by Jacques Patarin [Pa96].

This text is organised as follows: this section provides an overview about public key cryptography while the following section describes how to use HFE for encryption/decryption and signature generation/verification. Section 3 describes some attacks against the HFE system, and Section 4 illustrates how HFE can be modified to counter these attacks. In Section 5, we consider applications of HFE. This thesis concludes with Section 6. The whole text is based on a project at University College Cork entitled “Hidden Field Equations (HFE)” [Wo01]. Its aim was to implement a version of this cryptographic system in Java and it took place under the supervision of Patrick Fitzpatrick and Simon N. Foley.

The meaning of symbols is explained in Figure 1.1.

$a \stackrel{?}{=} b$	compare if $a = b$ is satisfied for given a, b
$\mathbb{N}, \mathbb{N}_0, \mathbb{Z}$	positive integers, non-negative integers, integers
$\mathbb{Z}_n$	$:= \{0, \ldots, (n-1)\}$, i.e., integers modulo n
$\mathbb{Z}_n^*$	$:= \{x \in \mathbb{Z} \mid 0 \leq x \leq n, \gcd(x, n) = 1\}$, i.e., multiplicative group modulo n
$\mathbb{R}, \mathbb{C}$	real numbers, complex numbers
$\mathbb{F}$	a finite field
$\mathbb{F}_q$, GF(q)	finite field with q elements
$\mathbb{F}[t]$	polynomials in t with coefficients in $\mathbb{F}$
$\mathbb{F}^n$	n dimensional vector-space over $\mathbb{F}$
$\mathbb{F}^{n \times n}$	matrix over $\mathbb{F}$ with n rows and n columns
$\mathbb{A}_n(\mathbb{F})$	set of affine transformations $\mathbb{F}^n \to \mathbb{F}^n$
$\mathbb{F}[t]/i(t)$	field generated by the irreducible polynomial $i(t) \in \mathbb{F}[t]$
$a \in_R A$	random choice of a with equal probability for all elements from the Set A
A $\leq_{\text{poly}}$ B	problem A is reduced in polynomial time to problem B
∂p	degree of polynomial p
S_n	permutation in n objects, i.e., all n tuples from $\mathbb{N}^n$ which are permutations in n elements
$\lfloor a \rfloor$	$:= max\{x \in \mathbb{N} \mid x \leq a\}$ for $a \in \mathbb{R}$
$\lceil a \rceil$	$:= min\{x \in \mathbb{N} \mid x \geq a\}$ for $a \in \mathbb{R}$
$a \mid\mid b$	concatenation of the two strings a and b

Figure 1.1: Meaning of Symbols in this Thesis

1.1 Public Key Cryptography

A milestone in modern cryptography was the paper [DiHe76] from Diffie and Hellman about public key cryptography. In 1976, they introduced the idea of not using one single, secret key for both encryption and decryption but one key for encryption (public key, denoted k) and one key for decryption (private key, denoted K). This opened the door to a completely new type of cryptography and helped to overcome the serious problem of distributing keys. In addition, it made applications like digital signature possible.

To explain the overall idea of public key cryptography, we make use of the three characters Alice, Bob and Eve. In this setting, Alice wants to send a message to Bob and Eve wants to eavesdrop.

We assume that everybody knows Bob's public key k_b but only Bob knows his private key K_B. When Alice wants to send a message m to Bob, she performs the Function $m' := E(k_b, m)$ to encrypt the Message m. Here $E(\cdot,\cdot)$ and k_b are publicly known. Moreover, the computation of $E(\cdot,\cdot)$ is feasible. Therefore Alice can perform $E(k_b, m)$ to obtain m'. On the other hand, the Function $D(\cdot,\cdot)$ is publicly known, too, but not Bob's private key K_B. So only Bob can compute $m = D(K_B, m')$ and thereby obtain the original message m.

It is necessary to postulate for all given public/private key pairs (k_x, K_X) and all possible messages m that the decryption function reverses the encryption function, i.e., $m = D(K_X, E(k_x, m)$. Additionally, we assume that it is not feasible to compute K_X knowing k_x, or m knowing m' in this scheme.

The system described above can be used for encryption. Assuming that $E(\cdot,\cdot)$ and $D(\cdot,\cdot)$ commute, i.e., $m = E(k_x, D(K_X, m))$ also holds for any pair (k_x, K_X) and any message m, this system can be also used for signature. For example, the RSA (Rivest, Shamir, Adleman) system is commutative (see Section 1.2). But not all public key schemes share this property, e.g., for ElGamal (see Section 1.3), signature and encryption are obtained using two different functions while still using the same underlying problem. Interestingly, there are currently more schemes that can be used for signature than there are for encryption [Pa96]. One example of a *signature only* system is the graph isomorphism problem combined with the Hamilton cycle problem [Sc96, Sec. 5.1].

In the following sections we will have a look at RSA and ElGamal. Both can be used for encryption and signature.

1.2 Factorisation

One of the first public key schemes was RSA. It exploits, firstly, the fact that multiplying two integers is easy but factoring their product is difficult and, secondly, the Fermat-Euler theorem, i.e., $a^{\phi(n)} \equiv 1 \pmod{n}\ \forall a \in \mathbb{Z}_n^*, n \in \mathbb{Z}$ [Me+96, Sec. 2.4.3].
These two facts are used in the following key generation algorithm:

1. Generate $p, q \in_R \mathbb{Z}$ prime
2. Compute $n := pq$ and $\phi = (p-1)(q-1)$
3. Select $e \in \mathbb{Z}$: $\gcd(\phi, e) = 1$
4. Compute $d \in \mathbb{Z}$ with $de \equiv 1 \pmod{\phi}$

In Step (1), $\in_R$ denotes a random choice for p, q within the set $\mathbb{Z}$. For RSA, the key pair is:

- Public Key $k := (e, n)$
- Private Key $K := (d, n)$

The encryption can be done by computing $m' := E(k, m) := m^e \pmod{n}$ only using publicly known information. The decryption makes use of the private key $K = (d, n)$ and thereby computes

$$D(K, m') := (m')^d = m^{ed} = m^{1+l\phi} \equiv 1 \cdot m \equiv m \pmod{n} \text{ for some } l \in \mathbb{Z}$$

This equation holds as $ed \equiv 1 \pmod{\phi} \Rightarrow ed = 1 + l\phi$ for some $l \in \mathbb{Z}$. Moreover, as the public and the private key are "symmetric" in a sense that there is no difference in either choosing e and then computing d or first selecting d and then computing e, so is also the encryption / decryption process. The same function can be used for both purposes, just by changing the key to k or K. In addition, $E(\cdot, \cdot)$ and $D(\cdot, \cdot)$ commute as

$$E(e, D(d, m)) = (m^d)^e \equiv m^{de} \equiv m^{ed} \equiv (m^e)^d = D(d, E(e, m)) \pmod{n}$$

If n can be factored, i.e., for a given n it is possible to obtain its prime factors

$$p_1, \ldots, p_l \in \mathbb{N} : \prod_{i=1}^{l} p_i = n$$

the function $\phi(n)$ can be computed (see above for the case of two prime factors p, q of n) and hence d for a given public key $k = (e, n)$. This is the reason why RSA is said to rely on the factorisation problem. It is widely believed that the factorisation problem for large $n \in \mathbb{N}$ (e.g., 1024 bits) is computationally infeasible [Me+96, Sec. 8.2.3].

The contrary, namely that breaking RSA also means that the factorisation problem is solved, is an open problem. There might be ways to break RSA without solving the factorisation problem. As an overall conclusion, RSA is believed to be secure although there is no formal proof that it is equivalent to the factorisation problem. [Me+96, Sec 3.3]

1.3 Discrete Logarithm

ElGamal [Me+96, pp 294–296] is another important system that is widely used for public key cryptography. It uses the fact that solving discrete logarithms, i.e., solving $a^x \equiv b \pmod{n}$ for given $a, b, n \in \mathbb{Z}$, is not feasible for large n.
Key generation works as follows:

1. Generate prime $p \in \mathbb{Z}$
2. Find a generator $\alpha \in_R \mathbb{Z}_p$ for $\mathbb{Z}_p^*$
3. Select a random integer $a \in_R \{1, \ldots, p-2\}$
4. Compute $\alpha^a \bmod p$

Here, the key pair is:

- Public Key $k := (\alpha, \alpha^a, p)$
- Private Key $K := (\alpha, a, p)$

To perform encryption, compute

$$m' := E(m, k) := (\gamma, \delta) := (\alpha^l \mod p, m\alpha^{al})$$

where $l \in_R \{1, \ldots, p-2\}$ denotes a randomly selected integer. The message m can be revealed using the function

$$D(K, m') := (\gamma^{-1})^a \cdot \delta \equiv \gamma^{-a} \cdot \delta \equiv \alpha^{-al} m \alpha^{al} \equiv m \pmod{p}$$

Decryption requires the knowledge of the integer a. If it is not practical to solve $\alpha^a \equiv \alpha \pmod{p}$, ElGamal is secure. Thus, ElGamal relies on the discrete logarithm problem. On the other hand, to break ElGamal it may not be necessary to solve the discrete logarithm problem.

The corresponding signature algorithm [Me+96, pp.454–456] uses the same public / private key pair. Using the private key $K_B = (\alpha, a, p)$ during the computation and a hash function $h(\cdot)$ in step (4), Bob computes his signature as follows:

1. Select a random integer $l \in_R \{1, \ldots, p-2\}$ with $\gcd(l, p-1) = 1$
2. Compute $r := \alpha^l \mod p$
3. Compute $l^{-1} \mod (p-1)$

4. Compute $s := l^{-1}(h(m) - ar) \mod (p-1)$

The signature for a given message m is the pair (r, s).
To verify this signature, Alice uses the following equation:

$$\begin{aligned}
(\alpha^a)^r r^s &\equiv (\alpha^a)^{\alpha^l} (\alpha^l)^{l^{-1}(h(m)-a\alpha^l)} \\
&\equiv (\alpha^a)^{\alpha^l} \alpha^{ll^{-1}(h(m)-a\alpha^l)} \\
&\equiv \alpha^{a\alpha^l} \alpha^{h(m)-a\alpha^l} \\
&\equiv \alpha^{a\alpha^l - a\alpha^l} \alpha^{h(m)} \\
&\equiv \alpha^{h(m)}
\end{aligned}$$

So Alice computes $(\alpha^a)^r r^s$ and $\alpha^{h(m)}$ to verify Bob's signature. If the two numbers are equal, the signature is valid. If not, she will reject the signature. A variation of ElGamal with a special parameter choice and a different signature verification method is used as the US "digital signature algorithm (DSA)" [Me+96, pp. 452–454].

Elliptic curve cryptography makes also use of the discrete logarithm problem but uses a different group, namely a group generated by an elliptic curve [Me+96, BSS99].

After this concise overview of public key cryptography, we will now concentrate on Hidden Field Equations (HFE) and see how it can be used not only for encryption but also for signature. In addition, we consider attacks on HFE.

Chapter 2

General Hidden Field Equations

While the Introduction gave an overview of public key cryptography, this section will concentrate on the mathematical background of HFE and show how it can be used for both encryption and signature.

2.1 Trapdoor in HFE

The underlying problem for HFE, namely MQ, is $\mathcal{NP}$-complete over finite fields (see Section 3.2). The name MQ stands for "**M**ultivariable **Q**uadratic" and refers to the fact that the public key can be seen as a system of quadratic equations in many variables. In this section, we will see how HFE is a trapdoor for the corresponding MQ problem.

In HFE, a finite field $\mathbb{F}$ with $q := |\mathbb{F}|$ elements, characteristic char $\mathbb{F}$ (a prime) and an extension field $\mathbb{E}$ over $\mathbb{F}$ are used. The extension field is generated by the irreducible polynomial $i(t)$ over $\mathbb{F}$. This polynomial $i(t)$ has degree $n := \partial i(t)$. The extension field $\mathbb{E}$ is also identified with the vector space $\mathbb{F}[t]/i(t)$ and some operations of HFE are not performed in the field $\mathbb{E}$ but in the vector space $\mathbb{F}[t]/i(t)$. This means that every $e \in \mathbb{E}$ can be seen as a vector $(e_1, \ldots, e_n) : e_i \in \mathbb{F}$ and, moreover, as a polynomial of degree $n-1$ at most in $\mathbb{F}[t]/i(t)$, where addition is normal polynomial addition and multiplication is done modulo the irreducible polynomial $i(t)$.

The three secret parameters in HFE (i.e., its private key) are its two affine transformations $S, T : \mathbb{F}^n \rightarrow \mathbb{F}^n$ and its private polynomial $P : \mathbb{E} \rightarrow \mathbb{E}$, so the private key K is the triple $(S, P, T) \in \mathbb{A}_n(\mathbb{F}) \times \mathbb{E}[x] \times \mathbb{A}_n(\mathbb{F})$. The public key k are polynomials $(p_1, \ldots, p_n)$ over $\mathbb{F}$ where each of them depends on n

variables $(x_1, \ldots, x_n)$. How these public polynomials can be obtained from a given private key is explained in Section 2.3.

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

$$\begin{aligned}
P &: \mathbb{E} \to \mathbb{E} \\
P(x) &= \sum_i c^{(i)} x^{h_i}, \text{where} \\
& \quad c^{(i)} \in \mathbb{E}, h_i \leq d, \\
& \quad h_i \neq h_j : i \neq j, \\
& \quad h_i = \begin{cases} 0, & & \text{(this is the constant term)} \\ q^a, & a \in \mathbb{N}_0 & \text{(these are the linear factors)} \\ q^b + q^c, & b, c \in \mathbb{N}_0 & \text{(these are the quadratic factors)} \end{cases}
\end{aligned}$$

The purpose of this restriction on the powers h_i is to keep the public polynomials $(p_1, \ldots, p_n)$ "small", i.e., at most quadratic in $(x_1, \ldots, x_n)$. As we will see in Section 3.2, being quadratic is enough for a system of equations over a finite fields to be $\mathcal{NP}$-complete. On the other hand, as $x^q \to x$ is a linear transformation in $\mathbb{F} = \mathrm{GF}(q)$, $x^{q^b+q^c}$ can be expressed in terms of two linear transformations and hence as a transformation which is at most quadratic over $\mathrm{GF}(q)$.

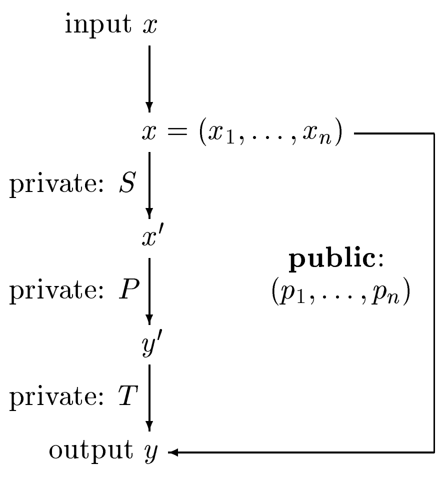

Figure 2.1: MQ trapdoor (S, P, T) in HFE

The two affine transformations S and T can be represented by one $n \times n$ matrix and one n-dimensional vector each. For example, the affine transformation $S \in \mathbb{A}_n(\mathbb{F})$ can be written as $S(x) = M_S x^t + v_s$ for some matrix $M_S \in \mathbb{F}^{n \times n}$, some vector $v_s \in \mathbb{F}^n$ and the vector $x = (x_1, \ldots, x_n)$.

As we see in Figure 2.1, 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 polynomial P can be inverted while there are no efficient algorithms available for inverting the public key [Pa96, Co01]. As the use of this trapdoor is slightly different for decryption (Section 2.2) and signature generation (Section 2.4), we will discuss its usage in detail in the corresponding sections. This section will be concluded with an observation on this trapdoor which apply to both decryption and signature generation.

In both cases, we have to transfer elements between $\mathbb{E}$ and $\mathbb{F}^n$ before we can apply the trapdoor (S, P, T): while the affine transformations S and T are applied to elements from $\mathbb{F}^n$, the polynomial P deals with elements from $\mathbb{E} = \mathbb{F}[t]/i(t)$. As $\mathbb{F}^n$ and $\mathbb{E}$ have the same number of elements, it is possible to construct a bijection between them. To do so, we observe how elements from these two sets look:

$$u_n t^{n-1} + u_{n-1} t^{n-2} + \cdots + u_2 t + u_1, u_i \in \mathbb{F}$$
$$\text{and } (u_1, \ldots, u_n) \in \mathbb{F}^n$$

So a bijection between $\mathbb{E}$ and $\mathbb{F}^n$ can be constructed as a correspondence between corresponding coefficients.

2.2 Encryption and Decryption of Messages using the Private Key

After we inspected the trapdoor in the previous section, we will now see how this trapdoor can be used for encryption (see Figure 2.2).

To encrypt a message m, we transfer it to a vector $(x_1, \ldots, x_n)$ over $\mathbb{F}^n$. Applying the transformation S to the vector, yields the result x'. At this point, note that x' is transformed from $\mathbb{F}^n$ to $\mathbb{E}$ in order to apply the private polynomial $P \in \mathbb{E}[x]$, using a simple correspondence between the coefficients (see previous section). The result $y' := P(x)$ is an element of $\mathbb{E}$. Once again, y' is transformed to the vector $(y'_1, \ldots, y'_n)$, transformation T is applied which gives the result $y = (y_1, \ldots, y_n)$. In addition, the redundancy r is computed from the original message m. So the final output is (y, r). The added redundancy r will be discussed later in this section.

$$
\begin{tikzcd}
\textit{plaintext } m \arrow[d, "\textbf{side computation: } \text{redundancy } r"] \\
x = (x_1, \dots, x_n) \arrow[d, "\text{private: } S"'] \arrow[ddd, bend left=90, "\textbf{public}: (p_1, \dots, p_n)"] \\
x' \arrow[d, "\text{private: } P"'] \\
y' \arrow[d, "\text{private: } T"'] \\
y
\end{tikzcd}
$$

Figure 2.2: HFE for Encryption of Message m with Ciphertext (y, r)

To decrypt y, the above steps are done in reverse order. This is possible as the private key (S, P, T) is known. The crucial step in deciphering is not the inversion of S and T, but rather the computation of x' in the equation $P(x') = y'$. As the polynomial P has degree d, there are up to d different solutions for this equation [LiNi86, Pa96]. Addition of redundancy to the message makes it possible to select the right m from the set of solutions $\{S^{-1}(x'_1), \dots, S^{-1}(x'_{d'})\} : d' \leq d$. This redundancy is added at the first step (see Figure 2.2) and must be done by sender and recipient using the same function. [Da01, Sec. 2.3.3] estimates that we need ≈ 80 bits redundancy if we want to have unique deciphering with probability $\geq 2^{-80}$.

While [Pa96] suggests that any non-linear function can be used to add the redundancy r, e.g., randomly generated quadratic polynomials in $x_1, \dots, x_n$, recent attacks (see, e.g., [Co01] or Section 3) suggest that the number of variables n and the number of equations m should have the relation $m \leq n$. So using randomly generated polynomials of degree 2 to compute the redundancy r is not recommended. A better choice would be to use cryptographically secure hash functions like SHA-1 or MD5 (see [Me+96] for these and other hash functions) as it seems unlikely that they can be expressed in terms of quadratic, multivariate polynomials.

Example

The following example shows how to encrypt and decrypt a message in $\mathbb{F} = \mathrm{GF}(2)$ and $\mathbb{E} = \mathrm{GF}(4)$ generated by $i(\gamma) = \gamma^2 + \gamma + 1$. Moreover, for this example, let

$$M_S := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, v_s := \begin{pmatrix} 1 \\ 1 \end{pmatrix}, M_T := \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, v_t := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Set $S := (M_S, v_s)$, $T := (M_T, v_t)$ and $P(x) := x^2 + 1$ to obtain the private key $K := (S, P, T)$.

Encryption: We encrypt the message $m = (1, 0)^t$ over $\mathbb{F}_2$.

1. Apply S:

$$\begin{aligned} M_S m + v_s &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned}$$

2. Transfer the result from $\mathbb{F}^2$ to $\mathbb{E}$: $(0, 1)^t \rightarrow 1$.

3. Apply P: $1^2 + 1 = 1 + 1 = 0$

4. Transfer the result from $\mathbb{E}$ to $\mathbb{F}^2$: $0 \rightarrow (0, 0)^t$.

5. Apply T:

$$\begin{aligned} M_T (0, 0)^t + v_t &= \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned}$$

So, $(1, 0)^t \rightarrow (0, 1)^t$ in this example. We did not need additional redundancy as P in this example can be uniquely inverted.

Decryption: For decryption, we have to reverse each step:

1. Apply T^{-1}:

$$\begin{aligned} M_T^{-1} [(0, 1)^t + v_t] &= \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \left[\begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right] \\ &= \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \end{aligned}$$

2. Transfer the result from $\mathbb{F}^2$ to $\mathbb{E}$: $(0,0)^t \to 0$.

3. Solve $P(x) = 0$: By inspection of all possible values $0, 1, \gamma, \gamma^2$ we see that only $x = 1$ satisfies this equation:

   - $P(0) = 0^2 + 1 = 1$
   - $P(1) = 1^2 + 1 = 0$
   - $P(\gamma) = \gamma^2 + 1 \equiv \gamma$
   - $P(\gamma^2) = \gamma^4 + 1 \equiv \gamma + 1$

4. Transfer the result from $\mathbb{E}$ to $\mathbb{F}^2$: $1 \to (0,1)^t$.

5. Apply S^{-1}:

$$\begin{aligned} M_S^{-1}[(0,1)^t + v_s] &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \left[\begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right] \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{aligned}$$

So we obtain the original message $m = (1,0)^t$ after decryption with the private key $K = (S, P, T)$.

2.3 Public Key: Generation and Encryption

Recall that the public polynomials $p_1, \ldots, p_n$ must be computed in such a way that they can be used to apply the private key (S, P, T) to message m without revealing this private key.

To compute these public keys, two different methods can be used. We will first describe key generation by polynomial interpolation (e.g., described in [MaIm88]).

2.3.1 Generation by Polynomial Interpolation

The idea of polynomial interpolation is to express a private key $K = (S, P, T)$ in terms of quadratic polynomials and to use pairs $(x, y) \in \mathbb{F}^n \times \mathbb{F}^n$ with $y := T(P(S(x)))$ to obtain the correct values for the coefficients of these polynomials. The public key polynomials $p_1, \ldots, p_n$ we deal with in this section have the form:

$$\begin{aligned} p_1(x_1,\ldots,x_n) &= \alpha_1 + \sum_{1\le j\le n} \beta_{1,j}x_j + \sum_{1\le j\le k\le n} \gamma_{1,j,k}x_jx_k \\ &\vdots \\ p_n(x_1,\ldots,x_n) &= \alpha_n + \sum_{1\le j\le n} \beta_{n,j}x_j + \sum_{1\le j\le k\le n} \gamma_{n,j,k}x_jx_k \end{aligned}$$

We know that these polynomial exist due to $x^q = x$ in $\mathrm{GF}(q)$ and the special choice of P. As the next section gives a constructive proof of their existence, we will just quote [MaIm88, Pa96]. Note that in the case of $\mathrm{GF}(2)$, we have $j \neq k$, as $x_jx_j = x_j^2 = x_j$, so x_jx_j is a linear rather than a quadratic factor. We will treat this case later in this section.

For $\mathbb{F} \neq \mathbb{F}_2$, we have $n + n^2 + n\frac{n(n+1)}{2} = \frac{n(2+n(n+3))}{2}$ coefficients $\alpha_i, \beta_{i,j}$, $\gamma_{i,j,k} \in \mathbb{F}$. To interpolate these polynomials, we therefore need at least $\frac{n(n+3)}{2} + 1$ elements from $\mathbb{F}^n$.

To make the computations easier, we will use a special choice of elements from $\mathbb{F}^n$: let $\eta_0 \in \mathbb{F}^n$ being a 0 vector, let $\eta_j \in \mathbb{F}^n : 1 \le j \le$ being a vector with its j^{th} coefficient being 1, the others 0, and let $\eta_{j,k} \in \mathbb{F}^n : 1 \le j < k \le n$ being a vector with its j^{th} and k^{th} coefficient being 1 and the others 0.

Evaluating the function $T(P(S(\cdot)))$ yields vectors with the following coefficients $(1 \le i \le n)$:

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

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

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

Revisiting this computation, we need $1 + 2n + \frac{n(n-1)}{2} = \frac{n(n+3)}{2} + 1$ elements from $\mathbb{F}^n$, as predicted.

In case of $\mathrm{GF}(2)$, the computation is easier as x_ix_i is a linear factor. So $\gamma_{i,j,j} = 0$ and we do not need to evaluate $T(P(S(a\eta_j)))$. In addition, we do not have to store $\gamma_{i,j,j}$, so the number of coefficients drops by n^2 to $n+n^2+n\frac{n(n-1)}{2} = \frac{n(2+n(n+1))}{2}$. Hence we expect that we need $\frac{n(n+1)}{2}+1$ elements from $\mathbb{F}^n$ to interpolate these polynomials. As the adjusted algorithm above interpolates exactly $1+n+\frac{n(n-1)}{2}$ elements from $\mathbb{F}^n$, our expectation is matched.

So both for $\mathbb{F} = \mathrm{GF}(2)$ and $\mathbb{F} \neq \mathrm{GF}(2)$, we can compute the public polynomials using polynomial interpolation with $O(n^2)$ points. In the next section, we will see why these quadratic polynomials are enough to express $T(P(S(\cdot)))$ over $\mathbb{F}^n$.

2.3.2 Base Transformation

Instead of interpolating the public key polynomials $k = (p_1, \ldots, p_n)$, we can also compute them directly. Before we treat the general case, we use a toy example to illustrate this technique. In each step of the following example, we mimic the computation for elements from $\mathbb{E}$ and $\mathbb{F}^n$, respectively.

The parameters for this toy example are: $\mathbb{F} = \mathrm{GF}(2)$, $i(t) := t^3+t+1$, $\mathbb{E} = \mathbb{F}[t]/i(t)$, and $P(x) = x^3 + (t+1)x + 1$. Here $(t+1) \in \mathbb{E}$; as a 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 expressing S in terms of affine polynomials p_1, p_2, p_3, we obtain the vector:

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

This vector has polynomial coefficients. However, it 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 $\mathbb{E}$. This is similar to the step in HFE where we change from $\mathbb{F}^n$ to $\mathbb{E}$ by identifying corresponding coefficients. Using the intermediate result $\mathfrak{P}[t]$, 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 $i(t) := t^3+t+1$. The interesting part in this computation lies in the fact that the operation "power 2" does not affect the polynomials but only t. This is due to the fact that $(a+b)^q = a^q+b^q$ and $a^q = a$ for $a, b \in \mathrm{GF}(q)$ [LiNi86]. Furthermore,

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

In this step, we obtain quadratic terms as we have to multiply affine polynomials. The next step is due to the fact that x in $P(x)$ has a constant multiple $(t+1)$:

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

So as an overall result, we can compute $P(x) = x^3+(t+1)x+1$ in terms of $\mathfrak{P}[t]$ and denote the result with $\mathfrak{Q}[t]$:

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

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

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

The affine transformation T can now be applied by ordinary matrix multiplication and vector addition. This step yields the result $\mathfrak{R}(x_1, x_2, x_3)$:

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

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 $(x_1, x_2, x_3) \in \mathbb{F}^3$.

This computation also gives a constructive proof why n quadratic polynomials in n variables are sufficient to express any private key (S, P, T). The overall algorithm consists of the following steps:

1. Express S in terms of a vector of polynomials $\mathfrak{P}$.
2. Compute $\mathfrak{P}[t]^{q^i}$ for $i = 0, 1, \dots a$ with $q^a \leq d \leq q^{a+1}$.
3. Compute $\mathfrak{P}[t]^{q^k+q^l} = \mathfrak{P}^{q^k}\mathfrak{P}^{q^l}$ using the results of Step 2.
4. Compute $c^{(i)}\mathfrak{P}[t]^{h_i}$ using the results from Steps 2 and 3.
5. Add up the results of Step 4 and obtain $\mathfrak{Q}$
6. Applying T to $\mathfrak{Q}$ yields result $\mathfrak{R}$.

We will now inspect for each step, if we can always apply it for any given private key. To do so, we recall that each private key $K = (S, P, T)$ consists of two affine transformations, namely S and T. Both can be expressed in terms of one matrix and one vector each.

Step (1) For Step (1), we obtain a polynomial vector $\mathfrak{P}$ by applying the affine transformation S to the vector $(x_1, \ldots, x_n)$. Calling the corresponding matrix M_S and the corresponding vector v_s, we can express this as

$$\begin{aligned}
S(x) &:= M_S x + v_s \\
&= \begin{pmatrix} s_{1,1} & \ldots & s_{1,n} \\ \vdots & \ddots & \vdots \\ s_{n,1} & \ldots & s_{n,n} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} s_{1,0} \\ \vdots \\ s_{n,0} \end{pmatrix} = \\
&= \begin{pmatrix} s_{1,1}x_1 + \ldots + s_{1,n}x_n + s_{1,0} \\ \vdots \\ s_{n,1}x_1 + \ldots + s_{n,n}x_n + s_{n,0} \end{pmatrix} \\
&= \begin{pmatrix} p_1(x_1, \ldots, x_n) \\ \vdots \\ p_n(x_1, \ldots, n_n) \end{pmatrix} \\
&= \mathfrak{P}(x)
\end{aligned}$$

Using the above construction, we can express each transformation $S \in \mathbb{A}_n(\mathbb{F})$ in terms of a polynomial vector. Each vector consists of at most affine polynomials. In this notation, an affine polynomial has at most linear and constant terms, but no quadratic terms (e.g., $s_{1,1}x_1$ is a linear term, $s_{1,0}$ is a constant term, x_1x_2 is a quadratic term). So the first step of the above algorithm will lead for any given affine transformation $S \in \mathbb{A}_n(\mathbb{F})$ to one vector $\mathfrak{P}$ with n rows of at most affine polynomials in the variables $(x_1, \ldots, x_n)$.

Step (2) In this step, we compute the result of $\mathfrak{P}[t]^{q^i}$ for $i = 0, 1, \ldots a$ with $q^a \le d \le q^{a+1}$. Here the input $\mathfrak{P}[t]$ is a polynomial in t. The coefficients of $\mathfrak{P}[t]$ are at most affine polynomials in $(x_1, \ldots, x_n)$:

$$\begin{aligned}
\mathfrak{P}[t] &:= p_n t^{n-1} + \ldots + p_2 t + p_1 \\
p_i &:= \alpha_{i,0} + \alpha_{i,1}x_1 + \ldots + \alpha_{i,n}x_n \text{ for } \alpha_{i,j} \in \mathbb{F}
\end{aligned}$$

We want to deal with the question, which kind of polynomials we obtain performing the operation $\mathfrak{P}[t]^{q^i}$. So we recall that $x^q = x$ for $q = |\mathbb{F}|$, and also that $(a+b)^q = a^q + b^q : a, b \in \mathbb{F}$ [LiNi86]. Let $p = \alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n$

and $q = |\mathbb{F}|$. Then the following holds:

$$\begin{aligned}
p^q &= (\alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n)^q \\
&= \alpha_0^q + (\alpha_1 x_1)^q + \ldots + (\alpha_n x_n)^q \\
&= \alpha_0 + \alpha_1^q x_1^q + \ldots + \alpha_n^q x_n^q \\
&= \alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n \\
&= p
\end{aligned}$$

Hence raising p to the power of q does not affect the affine polynomial p, so p^q is the identity operation for affine polynomials over $\mathbb{F}$.

In addition, computing ap for constant $a \in \mathbb{F}$ and affine polynomial p will lead to another affine polynomial, denoted p':

$$\begin{aligned}
ap &= a(\alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n) \\
&= a\alpha_0 + a\alpha_1 x_1 + \ldots + a\alpha_n x_n \\
&= \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n \\
&= p'
\end{aligned}$$

for some $\beta_0, \ldots, \beta_n \in \mathbb{F}$ and some affine polynomial p'. Having these preliminary results, we see how to compute $\mathfrak{P}^q$:

$$\begin{aligned}
\mathfrak{P}[t]^q &= (p_n t^{n-1} + \ldots + p_2 t + p_1)^q \\
&= p_n^q t^{(n-1)q} + \ldots + p_2^q t^q + p_1^q \\
&= p_n t^{(n-1)q} + \ldots + p_2 t^q + p_1 \\
&\equiv p'_n t^{n-1} + \ldots + p'_2 t + p'_1 \text{ for some affine polynomials } p'_1, \ldots, p'_n
\end{aligned}$$

In this context, $\equiv$ denotes reduction using the irreducible polynomial $i(t)$. As this reduction step consists of adding multiples of affine polynomials, it also yields affine polynomials, denoted $p'_1, \ldots, p'_n$. Although $\mathfrak{P}[t]^q$ is not the identity operation, the degree of the coefficient polynomials $p_1, \ldots, p_n$ is one at most, so they are still affine in terms of $(x_1, \ldots, x_n)$.

Step (3) As we saw in the last section, all results $\mathfrak{P}[t]^{q^i}$ of Step (2) are affine in terms of the coefficient polynomials $p_1, \ldots, p_n$. In Step (3), when we compute $\mathfrak{Q}[t] = \mathfrak{P}[t]^{q^i}\mathfrak{P}[t]^{q^j}$, these coefficient polynomial $q_1, \ldots, q_n$ are no longer affine but quadratic (i.e., having terms of the form ax_1x_2 for $a \in \mathbb{F}$). We will see in this section that they are at most quadratic. Consider

$$\begin{aligned}
\mathfrak{P}[t]^{q^i} &= p_n t^{n-1} + \ldots + p_2 t + p_1 \\
\mathfrak{P}[t]^{q^j} &= p'_n t^{n-1} + \ldots + p'_2 t + p'_1
\end{aligned}$$

Multiplying these two polynomials yields

$$\begin{aligned}\mathfrak{P}[t]^{q^i}\mathfrak{P}[t]^{q^j} &= (p_n t^{n-1} + \ldots + p_2 t + p_1)(p'_n t^{n-1} + \ldots + p'_2 t + p'_1) \\ &= (p_n p'_n)t^{2n-2} + \ldots + (p_2 p'_1 + p_1 p'_2)t + (p_1 p'_1) \\ &\equiv q_n t^{n-1} + \ldots + q_2 t + q_1\end{aligned}$$

for some polynomials $q_1, \ldots, q_n$ over $\mathbb{F}$. The interesting step is multiplying two affine polynomials p_i, p'_j for $1 \leq i, j \leq n$. Writing

$$\begin{aligned}p_i &= \alpha_{i,0} + \alpha_{i,1}x_1 + \ldots + \alpha_{i,n}x_n \\ p'_j &= \beta_{j,0} + \beta_{j,1}x_1 + \ldots + \beta_{j,n}x_n\end{aligned}$$

for some $\alpha_{i,k}, \beta_{j,l} \in \mathbb{F}$, we can express the multiplication of two affine polynomials in a matrix like way (see Figure 2.3). So multiplying two affine polyno-

$q = p_i \times p'_j$	$\beta_{j,0}$	$\beta_{j,1}x_1$		$\beta_{j,n}x_n$
$\alpha_{i,0}$	$\alpha_{i,0}\beta_{j,0}$	$\alpha_{i,0}\beta_{j,1}x_1$	$\ldots$	$\alpha_{i,0}\beta_{j,n}x_n$
$\alpha_{i,1}x_1$	$\alpha_{i,1}\beta_{j,0}x_1$	$\alpha_{i,1}\beta_{j,1}x_1^2$		$\alpha_{i,1}\beta_{j,n}x_1x_n$
$\vdots$	$\vdots$		$\ddots$	$\vdots$
$\alpha_{i,n}x_n$	$\alpha_{i,n}\beta_{j,0}x_n$	$\alpha_{i,n}\beta_{j,1}x_1x_n$	$\ldots$	$\alpha_{i,n}\beta_{j,n}x_n^2$

Figure 2.3: Multiplication of Two Affine Polynomials p_i, p'_j

mials p, p' yields one at most quadratic polynomial q. Moreover, adding two at most quadratic polynomials will also yield one at most quadratic polynomial. As we saw for Step (2), applying the reduction operation $\equiv$ does not change the overall degree of polynomials. So the polynomials $q_1, \ldots, q_n$ are at most quadratic over $\mathbb{F}$. In addition, they have the same variables $(x_1, \ldots, x_n)$ as the original polynomials $p_1, \ldots, p_n, p'_1, \ldots, p'_n$.

Steps (4) and (5) For these steps, we multiply a polynomial $\mathfrak{Q}[t]$ with constants $c^{(i)} \in \mathbb{E}$. In terms of finite field $\mathbb{F}$, we can express both as

$$\begin{aligned}\mathfrak{Q}[t] &= q_n t^{n-1} + \ldots + q_2 t + q_1 \\ c^{(i)} &= \gamma_n t^{n-1} + \ldots + \gamma_2 + \gamma_1\end{aligned}$$

for at most quadratic polynomials $q_1, \ldots, q_n$ and coefficients $\gamma_1, \ldots, \gamma_n \in \mathbb{F}$. To mimic the multiplication operation $c^{(i)}\mathfrak{Q}[t]$, we have to apply multiplication and addition of elements of the finite field $\mathbb{F}$. As $\mathbb{F}$ is a field, the result is still in $\mathbb{F}$. The reduction operation will also yield coefficients from

$\mathbb{F}$. So for $\mathfrak{Q}[t]$ being a polynomial with at most affine coefficient polynomials $q_1, \ldots, q_n$, the result still has affine coefficient polynomials, while for $\mathfrak{Q}[t]$ having at most quadratic polynomial coefficients, the result also has at most quadratic polynomial coefficient. So Step (4) does not change the degree of the coefficient polynomials. This is also true for Step (5), i.e., for adding the results of Step (4).

Step (6) The last operation is to apply the affine transformation T. As for the affine transformation S, we can mimic this using a matrix-vector multiplication and a vector-vector addition. Let

$$\mathfrak{Q} = \begin{pmatrix} q_1(x_1, \ldots, x_n) \\ \vdots \\ q_n(x_1, \ldots, x_n) \end{pmatrix}$$

for some at most quadratic polynomials $q_1, \ldots, q_n$ in $(x_1, \ldots, x_n)$ over $\mathbb{F}$. So applying the affine transformation T - consisting of the matrix $M_T \in \mathbb{F}^{n \times n}$ and the vector $v_t \in \mathbb{F}^n$ - yields

$$\begin{aligned} T(\mathfrak{Q}) &= M_T \mathfrak{Q} + v_t \\ &= \begin{pmatrix} t_{1,1} & \ldots & t_{1,n} \\ \vdots & \ddots & \vdots \\ t_{n,1} & \ldots & t_{n,n} \end{pmatrix} \cdot \begin{pmatrix} q_1 \\ \vdots \\ q_n \end{pmatrix} + \begin{pmatrix} t_{1,0} \\ \vdots \\ t_{n,0} \end{pmatrix} = \\ &= \begin{pmatrix} t_{1,1} q_1(x_1, \ldots, x_n) + \ldots + t_{1,n} q_n(x_1, \ldots, x_n) + t_{1,0} \\ \vdots \\ t_{n,1} q_1(x_1, \ldots, x_n) + \ldots + t_{n,n} q_n(x_1, \ldots, x_n) + t_{n,0} \end{pmatrix} \\ &= \begin{pmatrix} r_1(x_1, \ldots, x_n) \\ \vdots \\ r_n(x_1, \ldots, n_n) \end{pmatrix} \\ &= \mathfrak{R} \end{aligned}$$

for some at most quadratic polynomials $r_1, \ldots, r_n$ and a vector $\mathfrak{R}$. As for the affine transformation S, the overall degree of the polynomials keeps untouched. So the result vector $\mathfrak{R}$ has coefficients with at most quadratic polynomials.

As we saw in this section, each step of HFE can be mimicked using multivariable polynomials. For the first two steps, these polynomials are affine. From Step (3) onwards, we obtain polynomials of degree 2 at most.

Due to the construction of the public key it "bypasses" the private key $K = (S, P, T)$. Due to the way addition and multiplication operations are mixed, it seems unlikely that the private key K can be revealed using the corresponding public key k. However, we will investigate this question more closely in Section 3.

2.3.3 Encryption

Having the public key $k = (p_1, \ldots, p_n)$, encryption is very easy: it only requires the evaluation of n polynomials in n variables. Assuming that all operations over $GF(q)$ can be done in constant time - which is certainly the case if q is small enough - we will need $O(n^3)$ operations to evaluate these n polynomials with $O(n^2)$ terms each. [MaIm88, CGP01] discuss how to implement this polynomial evaluation in an efficient way.

2.4 Message Signature

In addition to encryption / decryption, HFE can also be used for signing a message m. As for decryption, we assume that it is computationally not feasible to get a solution $(x_1, \ldots, x_n)$ for

$$\begin{aligned} y_1 &= p_1(x_1, \ldots, x_n) \\ y_2 &= p_2(x_1, \ldots, x_n) \\ &\vdots \\ y_n &= p_n(x_1, \ldots, x_n) \end{aligned}$$

when $(p_1, \ldots, p_n)$ are quadratic in $x_1, \ldots, x_n$ without the trapdoor (S, P, T) (see sections 3.2 and 2.1). Figure 2.4 follows this notation, so the input for signature generation is denoted y, while the output is called x. In addition, the message m consists of t elements from $\mathbb{F}$, i.e., $(m_1, \ldots m_t) \in \mathbb{F}^t$. The vector $(r_1, \ldots, r_f) \in_R \mathbb{F}^f$ is randomly chosen (see below).

By using the private key $k = (S, P, T)$, the problem of finding a solution x for given y, reduces to find a solution to the equation $P(x') = y'$ where the polynomial $P \in \mathbb{E}[x]$ has degree d. This is feasible. Unfortunately for HFE, $P(x')$ is usually not a surjection and therefore $\exists y' : P(x') \neq y' \; \forall x' \in \mathbb{E}$. Keeping this in mind, we cannot find a solution $(x_1, \ldots, x_n)$ for each MQ problem with HFE trapdoor. So from a practical point of view, if we do not succeed in finding a solution x' for a certain y' in $P(x') = y'$, we have to try another y' until we obtain a result x'. In HFE, the number of y' we have to

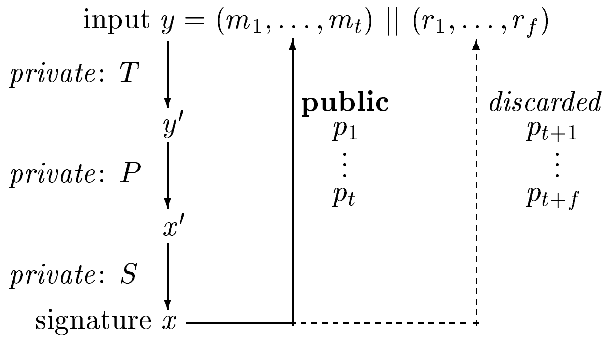

Figure 2.4: Signature with MQ, using the HFE trapdoor

try is small [Pa96]. For a special system such as Quartz (see Section 5.2), we expect to find a solution for one given y' with probability $1 - \frac{1}{e}$, i.e., approx. 60%. However, as Quartz tries up to 128 different y' for a given message, the overall probability for not finding any solution drops to approx. 2^{-185} and is therefore negligible [CGP01, p. 9].

For signature generation, we assume that the message $m \in \mathbb{F}^t$ and $n = t + f$. Here, $f \in \mathbb{N}$ are free input variables for a random string, i.e., different choices of y' for a given message. So $y = (m_1, \ldots, m_t)||(r_1, \ldots, r_f)$ where $\cdot||\cdot$ denotes the concatenation function and $(r_1, \ldots, r_f) \in_R \mathbb{F}^f$ is randomly chosen. The parameter f has to be chosen according to the field size of $\mathbb{F}$. As the parameters in the Quartz scheme are $\mathbb{F} = \mathrm{GF}(2)$, and $f = 7$, there are $2^7 = 128$ different y for each given message m. In general, we have q^f different y' for a given message m. If we can solve the corresponding $P(x') = y'$ for one of these q^f different y, we publish the corresponding $x = S^{-1}(x')$ as signature of m. See Figure 2.4 for the overall structure of a signature scheme.

Anybody who wants to verify that the message $m = (m_1, \ldots, m_t)$ was signed by the owner of the private key $K = (S, P, T)$ with $x = (x_1, \ldots, x_n)$, uses the public key, that is, $k = (p_1, \ldots, p_t)$ and compares (denoted $\stackrel{?}{=}$):

$$\begin{aligned} m_1 &\stackrel{?}{=} p_1(x_1, \ldots, x_n) \\ m_2 &\stackrel{?}{=} p_2(x_1, \ldots, x_n) \\ &\vdots \\ m_t &\stackrel{?}{=} p_t(x_1, \ldots, x_n) \end{aligned}$$

If all t equations are satisfied, the signature is valid. Otherwise, it is not. Note that only t of the $n = t + f$ public equations are necessary to verify a signature, the equations $p_{t+1}, \ldots, p_{t+f}$ are not used. Therefore in a signature scheme, only t equations will be published and f equations can be discarded. First, this will lead to a shorter public key. Second, as we will see in Section 4, this is also expected to improve the security of HFE.

2.5 Speed Considerations

As for any other algorithm, the question of speed is certainly a concern for HFE. The steps in HFE are:

1. Evaluate the public key $p_1, \ldots, p_n$ for given x.
2. Apply an affine transformation to a vector from $\mathbb{F}^n$.
3. Apply a polynomial to an element from $\mathbb{E}$.
4. Reverse an affine transformation for a vector from $\mathbb{F}^n$.
5. Solve the polynomial equation $P(x) = y$ for given $y \in \mathbb{E}$

Steps (1)-(4) are expected to be rather fast. In fact, in our Java implementation [Wo01] they take only milliseconds. The expensive part is Step (5). Both our own implementation [Wo01] and the NESSIE submission of Quartz [CGP01] suggest that this step will take several seconds. Having operations over $\mathbb{F}$ to be unit operations, [Pa96, Sec. 5.6] estimates its asymptotic complexity for this step to be $O(d^2n^3)$ or $O(dn^3 + d^3n^2)$ - depending on the chosen algorithm.

2.6 Security Parameters

This section summarises the security parameters of HFE.

Field Size q of Field $\mathbb{F}$

It is possible to implement HFE with any given finite field $\mathbb{F}$. However, it should be small for two reasons. First of all, this allows easier finite field operations, e.g. for signature verification and encryption. Secondly, to keep d small, as d is a function in $q^a + q^b$ for $a, b \in \mathbb{N}$. So increasing q results in an exponential growth of d. On the other hand, a smaller q will result in many coefficients for the public polynomials and therefore a bigger public key K. So this parameter has to be set in terms of a trade off between the speed of the signature on the one hand and the size of the public key on the other hand. From a practical point of view, only finite fields with characteristic 2 are of interest as they are easy to implement both in hard- and software.

Dimension n of the Extension Field $\mathbb{E}$

Increasing the parameter n will lead to a bigger extension field $\mathbb{E}$ and hence also to slower private key computation. In addition, the size of the public key will grow. However, [HFE] points out that increasing n is better than increasing d in terms of speed. In addition it recommends to have n being a prime to avoid the existence of subfields. Unless no attack is know which exploits the existence of subfields, this restriction is usually no problem and hence recommended.

Degree d of Polynomial P

While the private polynomial P must have a special form, namely only powers with the form $q^a + q^b$ for $a, b \in \mathbb{N}$, there is no further restriction on its degree d. However, having a small degree results in fast private key operations, in particular solving the equation $P(x) = y$ (see Section 2.5). In addition, simulations in [Pa96] suggest the form $d = 2a + 1$ for $a \in \mathbb{N}$ as there is a security "quantum leap" between $d = 2a$ and $d = 2a + 1$ but the private key operations have roughly the same speed.

2.7 Key Size

To give an idea how a special choice of the security parameters will lead to a particular key size, we will present a few figures for it. As you see in Figure 2.5, the size of the private key is negligible compared with the public key.

n	Public	Private
67	19kb	1.4kb
129	134kb	4.6kb
257	1Mb	17kb

Figure 2.5: Key Size in HFE with $q = 2$ and $d = 33$

Generally speaking, the public key size depends on the number of coefficients for the public polynomials and also on the number of elements in the finite field $q = |\mathbb{F}|$. As we saw in Section 2.3, we have to distinguish between the two cases $\mathbb{F} = \mathrm{GF}(2)$ and $\mathbb{F} \neq \mathrm{GF}(2)$. However, as we are mainly interested in the asymptotic size, we will only consider the case $\mathbb{F} \neq \mathrm{GF}(2)$. So the asymptotic size of the public key can be expressed as:

$$q \cdot n[1 + n + n(n-1)/2] = O(q \cdot n^3)$$

The private key size depends on the number of coefficients in the private polynomial P, the affine transformations S and T, and the number of elements in the finite field $q = |\mathbb{F}|$. We compute an upper bound that also depends on the degree d of P:

$$q[n(d+1) + 2 \cdot (n^2 + n)] = O(q \cdot (dn + n^2)) = O(q \cdot n(d+n))$$

So the size of the private polynomial P is negligible in comparison with the two affine transformations S and T. In addition, for q fixed, the public key will grow $O(n^3)$ while the private key will only grow $O(n^2)$.

2.8 Key Space

While the previous section was dealing with the storage size both for the private and the public key, the aim of this section is to compute the overall size of the key space for the private key. As the public key is derived from the private key, the overall number of public keys in HFE is either equal to or less than the size of the private key space. However, if two different private keys K_1, K_2 have the same public key k, these keys represent a trapdoor for the same HFE problem and are therefore functionally identical. As it is not clear a priori if or how many of such keys exist, all computations in this section are upper bounds.

Each private HFE key consists of two affine transformations $S, T \in \mathbb{A}_n(\mathbb{F})$ and one private polynomial $P \in \mathbb{E}[x]$. We start our estimation with two different upper bounds for the number of affine transformations in $\mathbb{A}_n(\mathbb{F})$. As each affine transformation $A \in \mathbb{A}_n(\mathbb{F})$ can be represented by a matrix $M_A \in \mathbb{F}^{n\times n}$ and a vector $v_a \in \mathbb{F}^n$, there are at most q^{n^2+n} such affine transformations for $q = |\mathbb{F}|$ and $n = \partial i(t)$. However, this also includes the case where the matrix M_A is not invertible and the corresponding transformation A is therefore not affine. To get a lower and more accurate upper bound, we compute the number of invertible matrices in $\mathbb{F}^{n\times n}$. Let $M_A \in \mathbb{F}^{n\times n}$ be an invertible matrix and $(m_a)_i \in \mathbb{F}^n$ be its i^{th} row vector. For the first vector $(m_a)_1$, we have $q^n - 1$ choices, as all vectors are allowed but the vector $(0, \ldots, 0)$. For the second vector $(m_a)_2$, we have $q^n - q$ possibilities, as the first and the second row have to be linearly independent (we want the matrix to be invertible). Another way of saying this is that we allow all choices for $(m_a)_2$ but the span of $(m_a)_1$, i.e., $\{f_1(m_a)_1 \mid f_1 \in \mathbb{F}\}$. With a similar argument, we obtain $q^n - q^2$ for $(m_a)_3$, as we now have to consider the span of $(m_a)_1$ and $(m_a)_2$, i.e., $\{f_1(m_a)_1 + f_2(m_a)_2 \mid f_1, f_2 \in \mathbb{F}\}$. Following this line, we obtain

$$\prod_{i=0}^{n-1} q^n - q^i$$

as the total number of invertible $n \times n$ matrices over $\mathbb{F}$.

Our next step is to compute the total number of private polynomials P for HFE. Here the number of coefficients depends on two conditions, namely the overall degree d of the private polynomial P and also the fact that each power in P has either the form 0, q^a or $q^b + q^c$ for some $a, b, c \in \mathbb{N}$. For given d and q, the number of coefficients λ can be computed as:

$$\lambda \;:=\; 1 + \lfloor \log_q d \rfloor + \sum_{i,j=0}^{q^i+q^j \leq d} 1$$

and the total number of HFE polynomials can be expressed as $q^{n\lambda}$.

Combining these results, we obtain $q^{n\lambda} \prod_{i=0}^{n-1} q^n - q^i$ as upper bound for the HFE key space. However, as this calculation also includes weak keys (e.g., S, T or P are the identity transformation) and no keys including the zero polynomial $0(x)$, the total number of keys is certainly lower. In fact, if there are private keys which yield the same public keys, i.e., if there are private keys K_1, K_2 which have different affine transformations S_1, S_2, T_1 or T_2 but are the trapdoor for the same public key k, the total size of the key space is certainly much lower. The author is not aware that this question has been investigated in detail in literature although a rigorous count of the total number of keys in the key space could lead to a deeper understanding of HFE and may yield a new kind of attack.

We will now compute the upper bound for the key space for a rather small HFE problem with the parameters $q = 2$, $n = 60$, $d = 17$. Assuming that this problem uses a polynomial with all possible coefficients, we obtain

$$\lambda = 1 + \lfloor \log_2 17 \rfloor + \sum_{i,j=0}^{2^i+2^j \leq 17} 1 = 1 + 4 + 7 = 12.$$

Using Maple, we compute

$$\prod_{i=0}^{59} 2^{60} - 2^i \approx 2^{3598}$$

So the size of the overall key space is at most

$$2^{60 \cdot 12} 2^{3598} = 2^{4318}$$

which is a rather big key space when compared to other public key cryptosystems. For example, in RSA a modulus size of 1024 or 2048 bits (see Section 1.2) is assumed to be sufficient, so one upper bound for the corresponding key space is 2^{1024} or 2^{2048}. However, all numbers in this section are only upper bounds. Making use of the internal structure of both key spaces (e.g., all moduli in RSA must be the product of exactly two different, rather big prime numbers), it is certainly possible to obtain a much smaller upper bound for the key space for both public key systems. However, having a big key space does not necessarily imply that the corresponding cryptographic system is secure. Therefore the next section deals with known attacks against HFE.

Chapter 3

Attacks on HFE

In the previous section, we were dealing with the so called "basic HFE" (short: HFE), i.e., HFE without any further variations, and described how it can be used to achieve an encryption or signature algorithm. In this section, we will look on vulnerabilities of HFE and hence how to attack it.

3.1 Types of Attacks

From a general point of view, there are two different problems for HFE and also two types of attack:

DEFINITION 3.1.1 *Given* (x, y) *a message/ciphertext or signature/message*[1] *pair and* k *a public key for HFE. Denote the following problems:*

1. *Inversion Problem: Recover* x *for given* y *and public key* k*.*
2. *Structural Cracking Problem: Recover the private key* $K = (S, P, T)$ *for given public key* k*.*

The attacks related to this two problems are called inversion attacks and structural attacks, respectively.

The notation in this definition mostly follows [Co01a]. The structural cracking problem is certainly more general than the inversion problem, as every attack which is able to reveal the private key K for a given public key k can be used to obtain the x for any given y. So every structural attack is also an inversion attack. This fact is pointed out in greater detail in [Co01a].

[1] As we denoted in Section 2.4 y the input and x the output for signature generation, (x, y) is indeed a signature/message pair.

Before dealing with the concrete attacks on HFE, we will first show how the underlying problem for HFE, namely MQ, is $\mathcal{NP}$-hard and in practice $\mathcal{NP}$-complete. So the inversion problem can be solved in $\mathcal{NP}$-time. In addition, we will show that the structural problem on HFE for given degree d can also be solved in $\mathcal{NP}$-time.

3.2 $\mathcal{NP}$-Completeness of MQ

MQ, i.e. solving quadratic equations in n variables over a given domain D, is the underlying problem for HFE. According to [GaJo79, p. 251], MQ over GF(2) is an $\mathcal{NP}$-complete problem. For the proof, it refers to a manuscript from A.S.Fraenkel and Y.Yesha that was unpublished in 1977 and also to "private communication" with L.G.Valliant. The text [FrYe79], from A.S.Fraenkel and Y.Yesha, was published in 1979, has the same title as the manuscript quoted in [GaJo79] and gives a proof for the $\mathcal{NP}$-completeness for polynomials over GF(2) and also the algebraic closure of GF(2). It mentions that strictly quadratic polynomials have been treated by L.G.Valiant, without giving any further hint how to prove this. It is called "MinRank" there and also in parts of the older literature as it was seen to be the minimal rank of a matrix. In the newer literature the same problem is usually denoted MQ (Multivariable Quadratic). This section is based on [PaGo97] which gives a proof both for the case of GF(2) and also for domains.

3.2.1 MQ over GF(2)

To show that MQ over GF(2) is $\mathcal{NP}$-complete, we will first show that it is in $\mathcal{NP}$ and then reduce 3-SAT to MQ.

DEFINITION 3.2.1 *MQ-GF(2): Consider m equations over GF(2) in n variables $x_1, \ldots, x_n$, also over GF(2), each equation consisting of quadratic terms (i.e., $x_i x_j$ but not $x_i x_j x_k$) at most. This is an instance of MQ over GF(2).*

Solvable in $\mathcal{NP}$-time The following $\mathcal{NP}$-algorithm solves MQ-GF(2):

1. Guess an assignment A for $(x_1, \ldots x_n) \in \{0,1\}^n$
2. Check if all m equations are satisfied by A.
3. Output `true` or go to an infinity loop, respectively.

As there are m equations, and each equation has $1+n+\frac{n(n-1)}{2}=\frac{n(n+1)}{2}+1$ terms at most, Step (2) requires polynomial time. If Step (2) is successful, the algorithm outputs `true`, terminates and otherwise goes to an infinity loop. So MQ-GF(2) can be solved in $\mathcal{NP}$-time.

Remark Note that in the case $\mathbb{F} = \mathrm{GF}(2)$, $x^2 = x\ \forall x$. This means that we can have $x_i x_j : i \neq j$ but not $x_i x_i$ as $x_i x_i = x_i^2 = x_i\ \forall i \in \{1, \ldots, n\}$ and x_i over GF(2).

$\mathcal{NP}$-hardness In this section, we will reduce 3-SAT to MQ-GF(2).

DEFINITION 3.2.2 *3-SAT: Let $B = \{b_1, \ldots, b_n\}$ be a set of boolean variables, let $L = \{b_1, \overline{b_1}, \ldots, b_n, \overline{b_n}\}$ be the corresponding set of literals, let $c_i \in (L \cup L^2 \cup L^3)$ be clauses of 3 literals at most, and let $C = \{c_1, \ldots, c_m\}$ be a set of these clauses. Then the corresponding 3-SAT problem is to determine if there is an assignment $A \in \{0,1\}^n$ for B such that all c_i are true and hence C is satisfied.*

We start with an instance of 3-SAT. Introduce $X = \{x_1, \ldots, x_n\}$ with x_i being variables over GF(2). Here $\vee$ denotes Boolean `OR` function and $+$ denotes addition over GF(2). Transfer each clause c_i to equation e_i using the following syntactical transformations where $l_i, l_j, l_k \in L$ and $b_i \in B$:

1. Replace $(l_i \vee l_j \vee l_k)$ by $(l_i + l_j + l_k + l_i l_j + l_i l_k + l_j l_k + l_i l_j l_k)$,
2. Replace $(l_i \vee l_j)$ by $(l_i + l_j + l_i l_j)$.
3. For each variable $b_i \in B$: replace $\overline{b_i}$ with $(1 - x_i)$ and b_i with x_i.
4. Construct an equation $e_i : (c_i' = 1)$ for each transformed clause c_i'.

This algorithm transfers each clause c_i into an equation e_i. Here all equations e_i have at most cubic terms. After expanding and collecting terms, we introduce $\frac{n(n-1)}{2}$ new variables $y_{i,j}$ and $\frac{n(n-1)}{2}$ new equations $y_{i,j} = x_i x_j$, for $i < j$. Moreover, we replace $x_i x_j$ or $x_j x_i$ by $y_{i,j}$ in all equations. This leads to $m + \frac{n(n-1)}{2}$ quadratic equations in $\frac{n(n+1)}{2}$ variables. If there is a solution for this set of equations, we also have a solution for the original 3-SAT problem. As all steps require only polynomial time and space, we reduced 3-SAT in polynomial time to MQ-GF(2), i.e., $\leq_{\text{poly}}$ MQ-GF(2).

Theorem 3.2.3 *MQ-GF(2) is $\mathcal{NP}$-complete.*

PROOF As shown above, MQ-GF(2) $\in \mathcal{NP}$ and 3-SAT $\leq_{\text{poly}}$ MQ-GF(2), which imply MQ-GF(2) is $\mathcal{NP}$-complete. □

3.2.2 MQ over Domains

In the previous section, we considered MQ over GF(2) and proved it to be $\mathcal{NP}$-complete. In this section, we will generalise this result for MQ over domains.

DEFINITION 3.2.4 *MQ-D: Let D be a domain, i.e., a commutative ring with 1 and without zero divisors. Then MQ-D is the problem of solving a set of m quadratic equations in n variables over D.*

$\mathcal{NP}$-hardness We will first reduce MQ-GF(2) to MQ-D, so consider the following set of m equations over GF(2):

$$\sum_{1\le i<j\le n} a_{i,j,k}x_ix_j + \sum_{i=1}^{n} b_{i,k}x_i + c_k = 0 \text{ for } (1 \le k \le m)$$

where $a_{i,j,k}, b_{i,k}, c_k \in \mathrm{GF}(2)$ are constants. These m equations form a MQ-GF(2) problem. We transfer each of these m equations to a set of $\frac{n(n+1)}{2}$ equations with $n + \frac{n(n-1)}{2} + (n-1) = \frac{n(n+3)-2}{2}$ variables:

$$\left\{\begin{array}{rcl}
a_{1,2,k}x_1x_2 & = & y_{1,2,k} \\
a_{1,3,k}x_1x_3 & = & y_{1,2,k} + y_{1,3,k} \\
& \vdots & \\
a_{n-1,n,k}x_{n-1}x_n & = & y_{n-2,n,k} + y_{n-1,n,k} \\
b_{1,k}x_1 & = & y_{n-1,n,k} + z_{1,k} \\
b_{2,k}x_2 & = & z_{1,k} + z_{2,k} \\
& \vdots & \\
b_{n-1,k}x_{n-1} & = & z_{n-1,k} + z_{n-1,k} \\
b_{n,k}x_n + c_k & = & z_{n-1,k}
\end{array}\right.$$

In this system of equations, $a_{i,j,k}, b_{i,k}, c_k, x_i$ are the same as above, while $y_{i,j,k}$ and $z_{i,k}$ are new variables over GF(2). So the whole system of m equations and in n variables x_i is transferred to a system of $\frac{mn(n+1)}{2}$ equations with $n + \frac{mn(n-1)}{2} + m(n-1) = \frac{2n+m(n-1)(n+2)}{2}$ variables. This step requires polynomial time.

After this initial step, we transfer the whole system from the finite field GF(2) to the domain D. This can be done by considering each $x_i, y_{i,j,k}, z_{i,k}$ as an element of D and by replacing each $a_{i,j,k}, b_{i,k}, c_k \in \mathrm{GF}(2)$ by the additive neutral $0 \in D$ or the multiplicative neutral $1 \in D$, respectively. During this transition, multiplication in GF(2) can be replaced by multiplication in D.

However, addition of two elements over GF(2) has to be replaced by the following term over D:

$$\begin{array}{rcl} \mathrm{GF}(2) & \to & D \\ (x+y) & \to & (x+y)((1+1)-(x+y)) \end{array}$$

where 1 denotes the multiplicative neutral in D. So for any $k \in 1, \ldots, n$ we obtain:

$$\left\{ \begin{array}{rcl} a_{1,2,k}x_1x_2 & = & y_{1,2,k} \\ a_{1,3,k}x_1x_3 & = & (y_{1,2,k}+y_{1,3,k})((1+1)-(y_{1,2,k}+y_{1,3,k})) \\ & \vdots & \\ a_{n-1,n,k}x_{n-1}x_n & = & (y_{n-2,n,k}+y_{n-1,n,k})((1+1)-(y_{n-2,n,k}+y_{n-1,n,k})) \\ b_{1,k}x_1 & = & (y_{n-1,n,k}+z_{1,k})((1+1)-(y_{n-1,n,k}+z_{1,k})) \\ b_{2,k}x_2 & = & (z_{1,k}+z_{2,k})((1+1)-(z_{1,k}+z_{2,k})) \\ & \vdots & \\ b_{n-1,k}x_{n-1} & = & (z_{n-1,k}+z_{n-1,k})((1+1)-(z_{n-1,k}+z_{n-1,k})) \\ z_{n-1,k} & = & (b_{n,k}x_n+c_k)((1+1)-(b_{n,k}x_n+c_k)) \end{array} \right.$$

In addition, we have to introduce $\frac{2n+m(n-1)(n+2)}{2}$ new equations (one for each variable) to make sure that we obtain a solution over D if and only if there is a solution over GF(2). Each of these equations has the form

$$x(1-x)=0.$$

So we transfered a system of m equations and in n variables over GF(2) to a system in $\frac{2n+m(n-1)(n+2)}{2}+\frac{mn(n+1)}{2}=n+m(n^2+n-1)$ equations and in $\frac{2n+m(n-1)(n+2)}{2}$ variables over D. All transformations can be done in polynomial time and space and the existence of a solution for the transferred problem implies the existence of a solution for the original one. Hence MQ-GF(2) $\leq_{\text{poly}}$ MQ-D for any domain D. So MQ-D is $\mathcal{NP}$-hard.

$\mathcal{NP}$-completeness Although [PaGo97] claims that MQ over *any* division ring is $\mathcal{NP}$-complete, we do not agree with this result. The reason is, that [PaGo97] omits to show that MQ-$D \in \mathcal{NP}$.

To show this, we consider the (slightly modified) algorithm from Section 3.2.1 for m equations in n variables over D:

1. Guess assignment $A \in R^n$ for $(x_1, \ldots x_n)$
2. Check if all m equations are satisfied by A.

3. Output `true` or go to an infinity loop, respectively.

The crucial part is Step (2): Although this checking can be done in polynomial time over GF(2), this is not true in general: consider the rings $\mathbb{R}$ and $\mathbb{C}$. Neither has zero divisors, both have a multiplicative neutral, and therefore the $\mathcal{NP}$-hardness proof from above applies. However, addition and multiplication in these two structures are not necessarily polynomial time operations and hence Step (2) can take more than $\mathcal{NP}$-time. So in general, MQ-D is $\mathcal{NP}$-hard. If all ring operations can be done in polynomial time, it is also $\mathcal{NP}$-complete.

Remark The question whether MQ-$\mathbb{Z} \in \mathcal{NP}$ requires further work. The key question is the number of bits which are needed to encode a single assignment A. Closely related is the status of MQ-$\mathbb{N}$. Although $\mathbb{N}$ is not a domain, it seems to be possible to transfer MQ-GF(2) to $\mathbb{N}$, using a similar algorithm as outlined in Section 3.2.2. However, as this thesis deals with using HFE for signature and encryption, and neither $\mathbb{Z}$ nor $\mathbb{N}$ seem to be of use for this aim, we will not investigate this question further.

3.2.3 Conclusions

In this section, we showed that MQ over GF(2) and also MQ over some domains is $\mathcal{NP}$-complete. This is especially the case when these domains are finite (and are fields, as every finite domain is also a field). However, for general domains MQ is $\mathcal{NP}$-hard, but not necessarily $\mathcal{NP}$-complete.

In general, being $\mathcal{NP}$-complete does not imply that a public key crypto-system using this problem is automatically secure. A counterexample is seen in crypto-systems using the knapsack problem. Although the knapsack problem is $\mathcal{NP}$-complete [GaJo79, p. 65], most of these crypto-systems were broken [Me+96, Sec. 8.6]. However, for the MQ problem, there is strong empirical and theoretical evidence [Pa96, HFE, SCPK00, Co01], that it is also hard on average (even when there is a HFE trapdoor) and hence can be used as basis for a secure public key crypto system.

However, as MQ-D where D is a finite domain is $\mathcal{NP}$-complete, the inversion problem for HFE can be solved in $\mathcal{NP}$-time.

3.3 $\mathcal{NP}$ structural attack

While the last section showed that the inversion problem for HFE can be solved in $\mathcal{NP}$-time, we will now show that for a given set of equations E and a given degree d it is possible to decide in $\mathcal{NP}$-time if a trapdoor of degree d exists or not.

The input is the set of equations E and the degree d. The critical part is d as it requires only $b := \lceil \log_2 d \rceil$ bits to express a given degree d while a polynomial P of degree d has up $d+1$ coefficients. So it looks as if the trapdoor (S, P, T) will grow exponentially with the input d. However, for HFE we concentrate on a more specific problem, namely trapdoors with specific values for the powers of x (see Section 2.1). As pointed out in this section, the Hamming weight for these powers is at most 2, hence the polynomial P has only $1 + b + \frac{b(b-1)}{2} = O(b^2)$ coefficients over $\mathrm{GF}(2^n)$. This is also true in the more general case where the equations E are defined over $\mathrm{GF}(q)$. So the trapdoor (S, P, T) can be expressed with a polynomial number of bits in terms of the input (E, d). After this observation, the $\mathcal{NP}$-time algorithm is simple:

1. Guess trapdoor $(S, P, T) \in \mathbb{A}_n(\mathbb{F}) \times \mathbb{E}[x] \times \mathbb{A}_n(\mathbb{F})$.
2. Transfer this trapdoor to a system of equations E', e.g., using one of the algorithms described in Section 2.3.
3. Compare the given system E with the new system E'. If they are equal, output `true`. Otherwise, go to an infinity loop.

As all steps can be done in $\mathcal{NP}$-time, the whole algorithm is in $\mathcal{NP}$. So deciding if a given system of equations E has a trapdoor of given degree d is in $\mathcal{NP}$.

Although this proof seems to imply that the problem of finding a trapdoor for a given system of equations E is in $\mathcal{NP}$, this is not true. The key point is that for a given set of equations, the corresponding polynomial P can have a very high degree d. When d grows exponentially, so will the number of coefficients of P and hence Step (1) requires exponential time and space. So finding an HFE trapdoor is in $\mathcal{NP}$, but not $\mathcal{NP}$-complete.

After these more complexity theoretical approaches, we will now concentrate on attacks which can be carried out in either polynomial or at least sub-exponential time.

3.4 Linear Attack

The linear attack described in this section is an inversion attack. Our description follows [Pa96]. For this special attack, we assume that we know y, y' and d where $y = HFE(x)$ and $y' = HFE(x+d)$. Here HFE is one invocation of the HFE system (either for signature or for encryption) as described in Section 2 and we know the difference d between the original messages $x, x+d$ but not x itself.

By calculating the difference between y and y' we can compute x as outlined below. We subtract two equations component-wise:

$$
\begin{aligned}
y_i - y_i' &= p_i(x_1, \ldots, x_n) - p_i(x_1 + d_1, \ldots, x_n + d_n) \\
&= (\alpha_{i,0} - \alpha_{i,0}) + \\
&\quad +\alpha_{i,1}(x_1 - x_1 - d_1) + \cdots + \alpha_{i,n}(x_n - x_n - d_n) + \\
&\quad +\alpha_{i,1,1}(x_1^2 - x_1^2 + 2x_1 d_1 - d_1^2) + \\
&\quad +\alpha_{i,1,2}(x_1 x_2 - x_1 x_2 - x_1 d_2 - x_2 d_1 - d_1 d_2) + \cdots + \\
&\quad +\alpha_{i,n,n}(x_n^2 - x_n^2 + 2x_n d_n - d_n^2) \\
&= \alpha_{i,1} d_1 + \cdots + a_{i,n} d_n + a_{i,1,1}(2x_1 d_1 - d_1^2) + \\
&\quad +\alpha_{i,1,2}(-x_1 d_2 - x_2 d_1 - d_1 d_2) + \cdots + \\
&\quad +\alpha_{i,n,n}(2x_n d_n - d_n^2)
\end{aligned}
$$

As all nonlinear factors of x_i vanish, this yields a system of linear equations in $x_1, \ldots, x_n$. A solution can therefore be computed in polynomial time, e.g., by Gaussian elimination. As this attack does not make use of the HFE trapdoor, it is a problem for any public key system which uses quadratic polynomials over a finite field as public key. In addition, it also works over a signature scheme, so keeping some public key polynomials secret does not solve this problem.

The linear attack can be avoided by padding the vector x with random elements from GF(q). In this case, their number has to be high enough to avoid exhaustive search for their values. Another way is to introduce a publicly known, linear attack resistant function $x = f(m)$, e.g. DES, AES with publicly known key k. This function $f(x)$ can then be applied to x before encrypting it [Pa96].

However, from a practical point of view, the linear attack is not too dangerous. As HFE can be used both for encryption and signature, we will consider both cases: in an encryption scheme, we usually do not encrypt a message m but a session key k_s, i.e., a random number which was generated with a cryptographically secure pseudo random number generator (CSPRNG). This CSPRNG does not allow to guess the difference d for two outputs x, x', so the linear attack should not be a threat. In a signature scheme, we will not deal with a message m, but rather with the hash of a message. Again, this hash will be computed using a cryptographically secure hash function. Moreover, as HFE is a non-linear function, two hashes h_1 and h_2 with $h_1 = h_2 + d$ will not have a similar relation between the corresponding signatures x_1 and x_2. So for a real world system, this linear attack should not be a threat.

3.5 Patarin Attacks

All Patarin attacks are an inversion attacks. We will first describe a simple version of the attack described in [Pa95] against the Matsumoto-Imai scheme from [MaIm88] and then illustrate how the more general attacks work against HFE itself.

3.5.1 Attack against Matsumoto-Imai

For Matsumoto-Imai, the main difference to HFE is the very restricted choice of P, namely, we allow only polynomials of the form $P(x) = x^{2^\theta+1}$ over $\mathrm{GF}(2^n)$ where $\theta \in \mathbb{N}$ and $2^\theta + 1$ coprime with $2^n - 1$. The reason for this choice is that such a P is a bijection and also allows fast secret key computations [MaIm88, Pa95, Pa96].

On the other hand, [Pa95] shows how to attack trapdoors of this type. We start with the equation $b = a^{2^\theta+1}$ over $\mathrm{GF}(2^n)$ where a is affine to x and b is affine to y. This is exactly HFE with a Matsumoto-Imai polynomial P and two affine transformations S, T. Now apply the operation $g : x \to x^{2^\theta-1}$ to both sides. This yields:

$$b^{2^\theta-1} = a^{2^{2^\theta}-1}$$

Multiply both sides by ab to obtain

$$ab^{2^\theta} = ba^{2^{2^\theta}}$$

which is linear in a and b in terms of the vector space $\mathbb{F}^n$ and $\mathbb{F} = \mathrm{GF}(2)$. Moreover, as a is affine in x and b is affine in y, we can express this equation in n equations

$$E_i : \sum_{1\leq j,k\leq n} \alpha_{i,j,k} x_j y_k + \sum_{j=1}^{n} \beta_{i,j} x_i + \sum_{j=1}^{n} \gamma_{i,j} + \delta_i = 0 \text{ with } 1 \leq i \leq n$$

over $\mathbb{F}^n$ in terms of $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$. This equation is quadratic in $x_j y_k$ and linear in $\alpha_{i,j,k}, \beta_{i,j}, \gamma_{i,j}, \delta_i$, so for given x and y, it can be computed by Gaussian reduction. Using the public key, we can compute y for a given $x \in \mathbb{F}^n$. So these equations E_i can be obtained easily from the public key by computing several pairs $(x, y) \in \mathbb{F}^n \times \mathbb{F}^n$ and then deducing the coefficients.

After these equations E_i have been computed, it is possible to obtain x for a given y as all equations are linear in x for given y. However, for a given y, we do not necessarily obtain a uniquely determined x as not all equations E_i are independent. Call the number of independent equations λ. If this number λ is small, the corresponding attack will reduce the search space only marginally. However, [Pa95, Thm. 6.3] shows that $\lambda \geq \frac{2n}{3}$ in all practical cases and $\lambda \geq n-1$ in most cases. So the search space is reduced by order of magnitude and hence Matsumoto-Imai polynomials are not secure for basic HFE.

3.5.2 Affine Multiple Attack

In [Pa96, Sec. 7], this attack is generalised further to the so called "affine multiple attack". The idea is, that for each polynomial P with degree d there is a least affine multiple $A(x, y)$ of $P(x) - y$. More formally:

DEFINITION 3.5.1 *Let $y \in \mathbb{E}$ and $P \in \mathbb{E}[x]$. The polynomial*

$$\sum_{i=0}^{t} \alpha_i x^{q^i} - y \textit{ where } t \in \mathbb{N}, \alpha_i \in \mathbb{E}$$

is called an affine polynomial over $\mathbb{E}$. If $A(x, y)$ is an affine polynomial over $\mathbb{E}$ and also a multiple of P then $A(x, y)$ is called an affine multiple of P. If $A(x, y)$ has minimal degree, we call it the least affine multiple of P.

The key point is that for any polynomial $P \in \mathbb{E}[x]$ there is an affine multiple with degree less than q^n. [Me+93, pp. 25-26] gives an constructive proof of this fact. As we see from Definition 3.5.1, all powers of x are linear in terms of $\mathbb{F}^n$. Using a similar argument as in Section 3.5.1, we can express this least affine multiple $A(x, y)$ in terms of $x, y \in \mathbb{F}^n$, even if x and y undergo an affine transformation (i.e., as we deal with HFE, the two affine transformations S and T). By evaluating "enough" pairs $(x, y) \in \mathbb{F}^n \times \mathbb{F}^n$, we can calculate the coefficients for these equations, as in Section 3.5.1. If "enough" is small, this will lead to an inversion attack against this special public key which is more efficient than exhaustive search on the message space.

In general, this attack is no threat to the security of HFE as the y terms are usually not linear in terms of $\mathbb{F}^n$. So their representation over $\mathbb{F}^n$ will lead to exponentially many terms for one given term over $\mathbb{E}$. However, if the powers of the y terms are bounded in terms of their Hamming weight, the polynomial will have only polynomial many coefficients in y^i or xy^j over $\mathbb{F}^n$. If w is the maximal Hamming weight of these exponents in y, the corresponding attack has to find the values for $O(n^{1+w})$ coefficients, as described in Section 3.5.1. As finding the correct values for these coefficients usually requires Gaussian reduction, it has complexity $O(n^{(1+w)\omega})$ at most, where $1 \leq \omega < 2.376$ is the complexity of the Gaussian reduction. [Pa96] justifies $\omega < 2.376$ as the obtained equations have a very special form. In either case, this attack is feasible for small values of the weight w.

In [Pa96], the affine multiple attack is applied against Dickson permutations, Dobbertin permutations, and many other polynomials. The permutations are especially vulnerable against this attack and hence not recommended for basic HFE. As an overall result, [Pa96] suggests choosing polynomials $P(x)$ which have many terms $\alpha x^{q^i+q^j}$, i.e., quadratic terms with nontrivial or better random coefficient $\alpha \in \mathbb{E}$.

3.5.3 Quadratic Attack

This attack is described in [Pa96, Sec. 8] and uses at most quadratic relations in x rather than affine relations in x. This looks strange at first glance as the security of HFE lies in the fact that it is not feasible to solve random polynomial equations with at least quadratic terms over any field for sufficiently many variables $x_1, \ldots, x_n$. However, if the system is largely overdefined, it becomes feasible. One possibility is to "linearise" the quadratic terms by introducing new variables $X_{i,j} := x_i x_j$ and solve this new system in $n + \frac{n(n-1)}{2}$ variables. By having λ independent equations and $\lambda > \frac{n(n-1)}{2}$, the corresponding vector space of solutions will be smaller than the vector space of messages. In Section 3.6, we will see other algorithms, which will also work for some $\lambda < \frac{n(n-1)}{2}$. Even on a second look, neither case seems to apply to HFE as the number of equations m and the number of variables n is equal for an encryption scheme and even $m < n$ for a signature scheme, so $\lambda \ll \frac{n(n-1)}{2}$.

However, these m equations may be not the only way of expressing the hidden private polynomial P over $\mathbb{F}$. If, additionally, these other equations are "preserved" by affine transformations, it is possible to use them for building more than these m equations in $x_1, \ldots, x_n$ where $x_i x_j$ is a term of highest degree. In this case "preserved" means not that an individual equation is invariant but their type, i.e. the degree of x, y does not grow by applying an affine transformation to them. [Pa96, Sec. 9] contains an exhaustive list of these equations and also a table which computes the number of such equations for the toy values $n = 17, 22$. Interestingly, the parameter n has no obvious effect but has d and also the number of coefficients for the given polynomial.

In the notation of [Pa96], the term $[XY]$ refers to equations which have quadratic terms only of the form $x_i y_j$, i.e.

$$[XY] := \left\{ \sum \alpha_{i,j} x_i y_j + \sum \beta_i x_i + \sum \gamma_i y_i + \delta_0 = 0 \;\middle|\; \alpha_{i,j}, \beta_i, \gamma_i, \delta_0 \in \mathbb{F} \right\}$$

So the quadratic attack is a further generalisation of the affine multiple attack from Section 3.5.2. Instead of looking for equations which are linear in x_i and having only a small number of coefficients in terms of y_j, this attack also allows factors $x_i x_j$, i.e., quadratic factors in x. The rest of this attack works the same way: evaluating pairs of $(x, y) \in \mathbb{F}^n \times \mathbb{F}^n$ and then finding the coefficients in these equations. After these equations are found, it is possible to compute x for given y.

The problem for this kind of attack is the higher number of equations needed to get x for a given y. Although the algorithms to be considered in Section 3.6 need fewer equations than linearization, the number is still $O(n^2)$. On the other hand, the simulations in [Pa96, Co01] show that many of these equations exist and only vanish gradually if the parameter d increases. For $d = 129$ and P having no special form, [Co01] indicates that there are no longer any useful equations.

However, again in [Co01], Nicholas Courtois shows that it is possible to work with "restricted" equations, i.e., equations with many x_i being fixed to 0, so the corresponding coefficients are no longer of concern. If the number of these x_i is chosen carefully, the equations which are obtained by this method can still be used to break basic HFE. [Co01] contains several improvements for this technique but concludes that basic HFE for $d > 128$ and $n > 80$ is secure and variations of HFE (see Section 4) are still unbroken. However, as [Co01] points out, the main problem for this kind of attack is not time but the memory which is needed to store the high number of coefficients.

3.6 Relinearization and beyond

The linearisation technique described in the previous section is able to solve m equations in n unknowns for $m \geq \frac{n(n-1)}{2}$. In this section, we will describe more recent improvements which also work for some $m < \frac{n(n-1)}{2}$. Due to time and space limitations, we only sketch these algorithms.

3.6.1 Relinearization

In [KiSh99], Kipnis and Shamir describe a technique called "relinearization". In brief, this technique is able to exploit equations like $(x_a x_b)(x_c x_d) = (x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c)$ (denoted "forth degree relinearization") and to solve m quadratic equations in n variables for $m \geq 0.1n^2$. In addition to [KiSh99, Sec. 5.2], this section is based on [SCPK00] and the far more detailed description in [Da01, Kap. 4.2.2]. As HFE uses at most quadratic equations, we will not consider extensions of the relinearization technique for higher degrees. So all systems of equations in n variables have at most quadratic terms $x_i x_j$ for $i < j$ over $\mathbb{F} = \mathrm{GF}(2)$ and $i \leq j$ over finite fields $\mathbb{F}$ with more than two elements. Denote $q = |\mathbb{F}|$ the number of elements in $\mathbb{F}$.

As in the linearization technique, the relinearization technique replaces quadratic terms like $x_i x_j$ with new variables $X_{i,j}$. As all non-linear terms are replaced, the new system can be solved with Gauss elimination. For less than $\frac{n(n-1)}{2}n^2 - n$ equations, this technique produces too many parasitic solutions, i.e., solutions which are only satisfying if we assume that the new variables $X_{i,j}$ are in fact independent from the original variables $x_i x_j$. To determine which of these solutions is correct, is is usually necessary to check all q^λ elements from an λ dimensional vector space over $\mathbb{F}$ where $\lambda \leq \frac{n(n-1)}{2}n^2 - n$. Kipnis and Shamir observed that equations like

$$(x_a x_b)(x_c x_d) = (x_a x_c)(x_b x_d) = (x_a x_d)(x_b x_c)$$

are trivially true from an algebraic point of view but are nevertheless linearly independent [KiSh99]. Therefore, these equations can be used to "filter" out parasitic solutions. [Da01, Kap. 4.2.2] gives a rigorous count of the number of different degree 4 equations. In Theorem 4.2.8 (forth degree relinearization), this number is proven to be

$$2\binom{n}{4} + 4\binom{n}{4} + \binom{n}{2} = \frac{1}{12}(n^4 - n^2).$$

As the terms $x_i x_j$ already have been replaced by $X_{i,j}$, the equation from above can be rephrased as $X_{a,b}X_{c,d} = X_{a,c}X_{b,c} = X_{a,d}X_{b,c}$. Introducing

λ new variables $\overline{X_1}, \ldots, \overline{X_\lambda}$, these $\approx \frac{1}{12}(n^4 - n^2)$ equations become linear again. An important question to answer is how many of these equations are in fact linearly independent. Experiments in [SCPK00] suggest that at least for large fields $\mathbb{F}$ these equations remain independent. So [SCPK00] and [Da01] conclude that forth degree relinearization can solve systems of $\geq 0.1n^2$ equations in n variables.

Degree 6 relinearization (or higher) is able to solve systems of ϵn^2 equations in n variables for $\epsilon < 0.1$. However, [SCPK00] and [Da01] point out that these higher degrees require far more computational power than degree 4 relinearization. In addition, there are far fewer independent equations of the form $(x_a x_b)(x_c x_d)(x_e x_f) = (x_{a'} x_{b'})(x_{c'} x_{d'})(x_{e'} x_{f'})$ for some $a, a', b, b', c, c', d, d', e, e', f, f' \in \mathbb{N}$ than expected in the first description of relinearization [KiSh99]. Both [SCPK00] and [Da01] show that Calay's theorem from group theorem applies, i.e., any permutation of indices can be expressed using permutations of at most two indices. For this and other reasons (see [SCPK00, Da01]), the number of independent equations drops dramatically and sixth degree relinearization is less useful than initially expected.

3.6.2 XL and FXL algorithm

In [SCPK00] another modification of the linearization technique, called "e**X**tended **L**inearization" or "multiplication (**X**) and **L**inearization" is introduced. It is proven to be at least as powerful as the relinearization technique [SCPK00, Sec. 8]. Again, [Da01, Kap. 4.2.3] gives a more detailed description of this method. Before describing the overall algorithm, we need to give some more definitions. Let $l_i : p_i(x_1, \ldots, x_n) - b_i = 0$ for $1 \leq i \leq m$ and given $b_i \in \mathbb{F}$ be m polynomial equations over $\mathbb{F}$ for p_i at most quadratic. Moreover, denote

$$\prod_{j=1}^{k} x_{i_j} l_i \text{ for } 1 \leq i \leq m \text{ and } i_1, \ldots, i_k \in \{1, \ldots, n\}$$

an equation of type $x^k 1$. As $a^q = a$ holds $\forall a \in \mathbb{F}$, all polynomials are reduced by $x_i^q \to x_i$. For some $D \in \mathbb{N}$, let I_D be the set of all equations the set $\{x^k l \mid 0 \leq k \leq D-2\}$ spans. Now the overall XL algorithm can be described as

1. **Multiply:** Generate all products $\prod_{j=1}^{k} x_{i_j} l_i \in I_D$ with $k \leq D-2$ for $1 \leq i \leq m$ and $i_1, \ldots, i_k \in \{1, \ldots, n\}$.

2. **Linearize:** Replace all terms obtained in Step 1 in more than one variable x_i with new variables. Eliminate these variables in a way that all terms containing one variable (e.g., x_1) are eliminated last.

3. **Solve:** Assume that Step 2 yields at least one univariate equation in the powers of x_1. Solve this equation over the finite field $\mathbb{F}$.

4. **Repeat:** Simplify the equations with the values obtained in Step 3 and repeat the process of finding the values of the other variables x_i.

If Step 3 is not successful, [Da01, Alg. 4.2.13] suggests to continue with a larger value of D.

Both [SCPK00, Da01] prove the correctness of the XL algorithm. The running time is estimated to be $O\left(\left(\frac{n^D}{D!}\right)^\omega\right)$ as the most time consuming step is the Gaussian elimination in $\approx \frac{n^D}{D!}$ variables (Step 2 of the XL algorithm). For usual Gaussian elimination, $\omega = 3$, and for improved versions of Gaussian elimination, $\omega \approx 2.3766$ [Da01, p. 109]

Unfortunately, the correct value D for given n, m is an open problem. [SCPK00] provides only experimental results. However, these result suggest that even for slightly overdefined systems (e.g., $m = n + 1$ or $m = n + 2$), the running time drops dramatically. Based on their simulations, [SCPK00, Sec. 6] expects

$$\begin{array}{lcl} m = n & : & D = 2^n \\ m = n + 1 & : & D = n \\ m \leq n + 2 & : & D = \sqrt{n} \end{array}$$

These observations lead to the so-called "FXL algorithm". The overall idea is to use an XL algorithm, but to embed it into a "guessing" scheme for few variables x_i. If the number of variables is small enough (e.g., two), this algorithm is expected to benefit from this huge gap between the running time for the case $m = n$ and $m \leq n + 2$. The drawback is an additional running time of $O(q^g)$ for g guessed variables.

Due to the huge gap in terms of running time between the cases $m = n$ and $m \leq n + 2$, it seems to be wise to avoid configurations with $n \geq m$ or even $n \ll m$ for HFE.

3.6.3 Gröbner Bases

In this section, we describe the two most recent attacks against HFE. Both are inversion attacks and use Gröbner Bases to solve the corresponding system of equations. Due to time and space limitations, we only sketch them.

The first is described in [CDF02] and makes use of the fact that a system of m equations in n variables over $\mathbb{F}$ is expected to have q^{m-n} solutions for $q = |\mathbb{F}|$ [CDF02, Sec. 3.2]. However, to forge a signature, we need only one of these solutions, so the Buchberger algorithm is doing far to much work in finding all of these q^{m-n} solutions. For $m > n$ it seems possible to fix some of the variables $x_1, \ldots, x_n$ to arbitrary values. When fixing f variables with correct values, we still expect q^{m-n-f} solutions on average. According to [CDF02, Sec. 3.2], fixing $f = m - n$ variables is the optimal choice as the system of equations is expected to have

$$q^{m-n-f} = q^{m-n-m+n} = q^0 = 1$$

solution on average. If the system has no solution at all, these f variables are fixed to other values from the underlying field $\mathbb{F}$. Their attack works against HFE itself and also against the modifications HFE- (see Sec. 4.5), HFEv (see Sec. 4.7) and their combination HFEv-. According to their experiments, l perturbations (e.g., l vinegar variables or l equations removed) lead to an extra security of $O(q^l)$ [CDF02, Sec. 6.4].

The second attack is due to Faugère and also uses Gröbner bases. In contrast to Courtois, Daum, and Felke, the attack of Faugère replaces the Buchberger algorithm with his own algorithm, called F5/2. Using this algorithm, he was able to break the HFE Challenge 1 in 96 hours on an AlphaServer DS20E (EV68 833 MHz) with 4 Gb of RAM [Fa02]. This challenge was proposed in 1996 by Jacques Patarin with the parameters $n = m = 80$ over GF(2). The degree of the secret polynomials is $d = 96$ [Pa96]. For breaking Challenge 1, Patarin has promised \$500. Faugère was able to compute all solutions of the corresponding system of equations [Fa02]. In [CDF02], the impact of this new algorithm on HFE is estimated. Their speed extrapolation suggest that the algorithm from Faugère in combination with their own variables fixing technique is the most powerful attack against HFE known so far. However, this attack has not been implemented yet.

3.7 Kipnis-Shamir Attack

The attacks discussed so far are inversion attacks against HFE. In contrast, the attack from Kipnis and Shamir is a structural attack. In fact, it is the only structural attack the author is aware of. To describe it, we use the original paper [KiSh99], [Da01, Kap. 5.2] and the improvements from [Co01, Sec. 8].

For their attack, Kipnis and Shamir do not consider HFE itself but a restricted version. For the remainder of this section, we therefore consider a restricted version of HFE. In this restricted version, the two affine transformations S and T are replaced by linear transformations, i.e., there are no vectors, only matrices. Moreover, the definition of the private polynomial P changes, too. It no longer has linear or constant terms and becomes

$$P(x) \quad = \quad \sum_{i,j=0}^{r-1} c^{i,j} x^{q^i+q^j}$$

where $c^{i,j} \in \mathbb{E}$ and $r \in \mathbb{N}$ is called the "rank" (see below). The degree d of this polynomial P is bounded: $d \leq 2q^{r-1}$. Therefore, in this section a public key $k = (p_1, \ldots, p_n)$ always represents a restricted private key $K = (S, P, T)$. Kipnis and Shamir justify this restriction: "the attacker can ignore lower degree monomials" [KiSh99, p. 4]. As the difficult part of HFE are the quadratic parts of the equations, we agree with this restriction.

Although the attack consists of several steps, two key ideas can be identified: first, the public key k over $\mathbb{F}$ can be expressed as a sparse polynomial equation in one variable x over $\mathbb{E}$. Second, the private polynomial P can be expressed in a non-standard form using a matrix with unusual small rank r.

The first step of this attack is described in lemmata 3.1, 3.2, and 3.3 in [KiSh99]. The result of Lemma 3.1 is that a linear mapping in n variables over $\mathbb{F}$ (e.g., S or T) can be expressed as a polynomial

$$y = \sum_{i=0}^{n-1} a_i x^{q^i} \text{ for } a_i \in \mathbb{E}$$

where y, x are expressed in a kind of "base" of $\mathbb{E}$ which corresponds to an arbitrary base of $\mathbb{F}^n$ (for a bijection between $\mathbb{F}^n$ and $\mathbb{E}$, see Section 2.1). While Lemma 3.1 uses a counting argument, Lemma 3.2 proves a similar result (replacing linear mappings with general polynomials) in a constructive way. Finally, Lemma 3.3 shows that these univariate polynomials in $\mathbb{E}$ are sparse, i.e., have at most $O(n^\delta)$ coefficients. In this notation, δ is the degree of the n polynomials in n variables. For example, a HFE public key has degree $\delta = 2$, and a linear mapping (e.g., S) has degree $\delta = 1$. If there were any term $\alpha_{a,b,c}x_a x_b x_c$ for $a \neq b$, $a \neq c$, $b \neq c$, and $\alpha_{a,b,c} \in \mathbb{F} \setminus \{0\}$ in these n polynomials, the degree δ would be 3.

Let $k = (p_1, \ldots, p_n)$ be a public key. As outlined in Section 2.3, the degree δ for these polynomials is 2. Using Lemma 3.3, we can represent the public key k over $\mathbb{E}$ as

$$G(x) = \sum_{i,j=0}^{n-1} g_{i,j} x^{q^i + q^j} = XGX^T$$

using the matrix $G = (g_{i,j}) \in \mathbb{E}^{n \times n}$, its coefficients $g_{i,j}$, the vector $X = (x^{q^0}, x^{q^1}, \ldots, x^{q^{n-1}})$, and its transpose. Note: this vector X is special, as its elements are related, not independent. In addition, XGX^T represents the public key k in non-standard form. In general there is no unique matrix G for a given public key k. Using the same trick, we can also express the private polynomial P, non-standard, as XUX^T for a matrix $U \in \mathbb{E}^{n \times n}$. Although this is clear from the definition of P, this matrix U is not unique. As outlined above, the two linear transformations S and T can be represented as univariate polynomials, denoted s, t. Using this univariate representation of the private key K, we can express the public key as a composition of the private key: $G(x) = t(U(s(x)))$. Rewriting this equation yields

$$t^{-1}(G(x)) = U(s(x))$$

with $s = \sum_{i=0}^{n-1} \sigma_i x^{q^i}$ and $t^{-1} = \sum_{j=0}^{n-1} \tau_j x^{q^j}$ for $\sigma_i, \tau_j \in \mathbb{E}$ (both s, t^{-1} represent linear mappings over $\mathbb{F}$).

In this equation, G is given and s, U, t are unknowns. To exploit their internal structure, we further rewrite this equation by introducing matrices G^{*l} and a matrix W. Here $W = (w_{i,j}) \in \mathbb{E}^{n \times n}$ with $w_{i,j} = (\sigma_{j-i})^{q^i}$ (with $(j-i)$ computed modulo n). In addition, the matrices G^{*l} (for $0 \leq l < n$) are derived from the matrix G by raising each of its entries to the power q^l and cyclically rotating forward both the rows and the columns of the result by l steps. We can write this using a simple recursion formula:

$$\begin{array}{rcll} G^{*0} & := & G & \\ G^{*l} & := & ((g^{*(l-1)}_{i-1,j-1})^q) & \text{for } 0 < l < n \text{ and} \\ & & & 0 \to n \text{ for the indices } i, j \end{array}$$

Moreover, [KiSh99, Thm. 4.1] proves that

- $P(S(x))$ is equal to WUW^T and

- $t^{-1}(G(x))$ can be expressed as $\sum_{l=0}^{n-1} \tau_l G^{*l}$.

So $\sum_{l=0}^{n-1} \tau_l G^{*l} = WUW^T$ holds. Defining $G' := \sum_{l=0}^{n-1} \tau_l G^{*l}$, we obtain the so-called "fundamental equation"

$$G' = WUW^T$$

As the public key $k = (p_1, \ldots, p_n)$ is given, its univariate representation G over $\mathbb{E}$ can efficiently be computed. In addition, the n matrices G^{*l} can be computed (see recursion formula, above). So G' is mostly known - apart from the coefficients $\tau_0, \ldots, \tau_{n-1}$. To obtain these n elements from $\mathbb{E}$, we exploit the fact that the degree d in a HFE system is rather small (e.g., 129 for Quartz, see Section 5.2). As $d \approx 2q^{r-1}$, the rank r is even smaller (at most 13, see [KiSh99]).

As $G' = WUW^T$, the rank of G' is also r, thus every $(r+1) \times (r+1)$ sub-matrix of G' has determinate 0. In total, there are $\binom{n}{r+1}^2$ such sub-matrices. According to [Co01, Sec. 8], we can assume that at least $\binom{n}{r+1}$ of the corresponding equations are linearly independent. Linearisation (see Section 3.5.3) allows to obtain solutions for this system of $\binom{n}{r+1}$ equations and finally yields the values of $\tau_0, \ldots, \tau_{n-1} \in \mathbb{E}$.

When the values $\tau_0, \ldots, \tau_{n-1}$, i.e., the linear transformation T, have been recovered, the next step in the Kipnis-Shamir attack is to find the linear transformation S, i.e., the values $\sigma_0, \ldots, \sigma_{n-1} \in \mathbb{E}$. To achieve this goal we assume, w.l.o.g., that the rank of the matrix U is exactly r and the rank of the matrix W is exactly n [KiSh99, Sec. 5.3]. We now define

$$\begin{aligned} \ker(U) &:= \{x \in \mathbb{E}^n \mid xU = 0\} \\ &= \{x = (x_1, \ldots, x_n) \in \mathbb{E}^n \mid x_i = 0 \text{ for } 1 \le i \le r\}. \end{aligned}$$

This equation holds as the rank of the matrix U is r and $u_{i,j} = 0$ for $i, j > r$. Exploiting $G' = WUW^T$ and also the fact that W (and W^T) are invertible, we obtain

$$\begin{aligned} \ker(G') &= \ker(WUW^T) \\ &= \{x \in \mathbb{E}^n \mid x(WU)W^T = 0\} \\ &= \{x \in \mathbb{E}^n \mid xWU = 0(W^T)^{-1}\} \\ &= \ker(WU) \end{aligned}$$

As the rank of G' is r, a basis $(v_1, \ldots, v_a)$ of $\ker(G')$ with $v_i \in \mathbb{E}^n$ has exactly

$a := n - r$ elements. Moreover, as

$$\begin{aligned}\ker(G') &= \ker(WU) \\ &= \{x \in \mathbb{E}^n \mid x(WU) = 0\} \\ &= \{x \in \mathbb{E}^n \mid (xW)U = 0\}\end{aligned}$$

$xW \in \ker(U)$ holds $\forall x \in \ker(G')$ and hence a base $(v_1, \ldots, v_{n-r})$ of the sub-space $\ker(G')$ yields vectors $(v_i W) \in \ker(U)$. As the kernel $\ker(U)$ has the special form

$$\{x = (x_1, \ldots, x_n) \in \mathbb{E}^n \mid x_i = 0 \text{ where } 1 \leq j \leq r\}$$

(see above), we obtain $(n-r)r$ equations in the coefficients $(v_i W)_j = 0$ over $\mathbb{E}$ for $1 \leq i \leq (n-r)$ and $1 \leq j \leq r$. As $W \in \mathbb{E}^{n \times n}$ and $r \ll n$, there are too few equations to determine all n^2 coefficients of the matrix W. Fortunately, due to its construction, the matrix W has a special form, namely all coefficients $w_{i,j} = \left((\sigma_{j-i})^{q^i}\right)$, so these n^2 variables over $\mathbb{E}$ can be expressed using n unknowns $\sigma_0, \ldots, \sigma_{n-1} \in \mathbb{E}$. The drawback of these equations are the very high degrees for the values σ_i. However, Kipnis and Shamir observed that these equations become linear over $\mathbb{F}$ as all the powers of σ_i have the form q^k for some $k \in \mathbb{N}_0$. Hence we are able to replace $(n-r)r$ non-linear equations in n unknowns over $\mathbb{E}$ by $n(n-r)r$ linear equations in n^2 unknowns over $\mathbb{F}$. For practical values of n, r (e.g., $n = 80$ and $r = 10$), this system is largely over-defined and therefore yields n^2 values $\sigma_{1,1}, \sigma_{1,2}, \ldots, \sigma_{n,n} \in \mathbb{F}$.

As the matrices G' and W are known, the equation $G' = WUW^T$ yields the matrix U. Using this matrix, the private polynomial P can be obtained and the whole private key $K = (S, P, T)$ is found.

Kipnis and Shamir outline in [KiSh99] that it is not clear a priori if there is a unique private key K for a given public key k. However, as we are only interested in finding a trapdoor (S, P, T), we are not concerned about the uniqueness of this solution. Any trapdoor will help to solve the corresponding MQ problem for given polynomials $k = (p_1, \ldots, p_n)$ over $\mathbb{F}$ and values $(y_1, \ldots, y_n) \in \mathbb{F}^n$.

Although this structural attack gives an algorithm to recover a private key $K = (S, P, T)$ for a given public key $k = (p_1, \ldots, p_n)$, [Co01, Sec. 10] shows that the time and space complexity is still higher than brute-force search within the message space $\mathbb{F}^n$ for any practical values of the HFE system.

Chapter 4

Variations

As we saw in the last section, there are various attacks available against HFE. These attacks both threaten the use of HFE for signature and for encryption. However, HFE is a quite flexible scheme, as it has different security parameters (see Section 2.6). In addition, we will see in this section that it can be changed in many ways. Some of them will enhance the security of HFE, others its speed, and some of them will be proven to have no positive effect on HFE. This section is based on [Pa96, Co01] except for sections 4.9 and 4.10. The modifications discussed there have been developed by the author. The author is not aware that they have been discussed elsewhere.

4.1 Bijection between $\mathbb{F}^n$ and $\mathbb{E}$ revisited

The bijection between $\mathbb{F}^n$ and $\mathbb{E}$ plays an important role in HFE, as it is frequently used in the trapdoor and also implicitly in the public key. This section will hence investigate the question if the security of HFE can be enhanced by the usage of another bijection but a simple correspondence between corresponding coefficients. On the first glance, any bijections can be used. However, only affine transformations are suitable for our purpose, as any other transformation cannot be expressed in terms of affine polynomials over $\mathbb{F}$, and will therefore lead to a larger key size. As the large public key size in HFE is already a problem (e.g., see [Pa96] and Section 2.7), we want its effect on the public key to be as small as possible. So only affine transformations are suitable for this purpose. Moreover, any affine transformation is suitable, as both S and T have the potential to express any affine transformation $\mathbb{F}^n \rightarrow \mathbb{F}^n$. Or rephrased: if we chose any affine transformation U as a bijection between $\mathbb{E}$ and $\mathbb{F}^n$, a special choice of S and

T will have the same effect as special choice of U. So from a crypto-graphical point of view, as every affine transformation occurs with equal probability, there is no need to have a very complicated one between $\mathbb{F}^n$ and $\mathbb{E}$ to enhance the security of HFE. And from a computational point of view, we want this transformation to be as easy as possible to save computational steps. So the bijection outlined in Section 2, i.e., a simple component-wise identification of elements from $\mathbb{F}$, is the best possible choice for our purpose. Consequently, [Pa96] does not discuss any other possibility but this bijection.

4.2 Working with a Subfield $\widetilde{\mathbb{F}}$ of $\mathbb{F}$

While the aim in the last section was to enhance security, we now want to concentrate on the size of the public key. To obtain a smaller public key, we choose both the coefficients in the two affine transformations S and T and also the coefficients in the private polynomial P in a way that it is possible to express the public key $k = (p_1, \ldots, p_n)$ in terms of a proper subfield of $\mathbb{F}$. For example, for $\mathbb{F} = \mathrm{GF}(256)$ and the subfield $\widetilde{\mathbb{F}} = \mathrm{GF}(2)$, we save $\frac{7}{8}$ of the size of the public key as each coefficient in the public polynomials $(p_1, \ldots, p_n)$ does no longer require 8 bits, but only 1 bit. However, the message space is still $\mathbb{F}^n = \mathrm{GF}(256^n)$ over the original finite field GF(256).

Unfortunately, this modification has two drawbacks. First of all, it leads to a big degree d, as $d = q^a + q^b$ for some $a, b \in \mathbb{N}$ grows exponentially with $q = |\mathbb{F}|$. Secondly, it reduces dramatically the key space for the private key. As [Pa96] points out, it is possible that an attacker could make use of this special structure of the public key. We will investigate this question further in Section 5.1.

4.3 Modifying the Private Polynomial

As we saw in Section 3, some private polynomials P are vulnerable to special kinds of attacks. It seems that all polynomials which are bijections on $\mathbb{E}$ are specially vulnerable. However, for a signature scheme, these polynomials have a very interesting property: as they are bijections, there is always a signature for any given message $m \in \mathbb{F}^n$. Although we saw a way in Section 2.4 to overcome this problem by adding some extra random elements $r_1, \ldots, r_f \in_R \mathbb{F}$ to the message m, it is time consuming as it needs - on average - several invocations of the root finding algorithm. So there is strong motivation for using these bijections as private polynomial P. Unfortunately, it is necessary to modify HFE so these bijections can be securely used. These modifications are discussed in the following sections.

As HFE makes use of at most quadratic polynomials, it is an interesting question what happens when they are replaced by polynomials of degree 3 at most. In terms of the private key, this means that we allow not only powers of the form $q^a + q^b$ for some $a, b \in \mathbb{N}$, but also $q^a + q^b + q^c$ for some $a, b, c \in \mathbb{N}$ for the private polynomial P'. In terms of private key operations, this does not significantly affect speed, as the number of coefficients of the private polynomial is not the problem for root finding, but its overall degree d (see Sec. 2.5). Although the key size of the private key grows, it is still tiny compared with the public key. For example, for $\mathbb{F}$ = GF(2) and $\mathbb{E}$ = GF(2^{129}), we obtain a public key of 134kb (see Section 2.7). A private polynomial with $d = 255$ has

$$\sum_{i=0}^{2} \binom{8}{i} = \binom{8}{0} + \binom{8}{1} + \binom{8}{2} = 1 + 8 + 28 = 37$$

coefficients. Without the two affine transformations S and T, we need $37 \cdot 129 = 4774$ bits, i.e., $\approx$ 600 bytes for this polynomial (assuming that all coefficients are from $\mathbb{E}$). For S and T, we need $2 \cdot ((129 \cdot 129) + 129) = 33,540$ bits, i.e., $\approx$ 4kByte. In comparison, we obtain

$$\binom{8}{3} = 56$$

additional coefficients for the private polynomial P' which also allows powers of the form $q^a + q^b + q^c$ for some $a, b, c \in \mathbb{N}$ and the overall degree d=255. This requires additional 7224 bits, i.e., $\approx$ 900 bytes for the private key, and is therefore negligible in comparison with the public key. However, the drawback is the key size of the public key. In our example, we obtain

$$1 + 129 + \frac{129 \cdot 128}{2} + \frac{129 \cdot 128 \cdot 127}{3!} = 357,890$$

coefficients per polynomial, i.e., $129 \cdot 357,890 = 46,167,810$ coefficients in total. Using one bit per coefficient, we need $\approx$ 5.5 mega byte for the public key. For an extension field of higher degree (e.g., $n = 257$), this figure rises further. In general: for at most quadratic polynomials, we have $O(n^2)$ coefficients for each polynomial. For polynomials of degree 3 at most, we obtain $1 + n + \frac{n(n-1)}{2} + \frac{n(n-1)(n-2)}{3!} = O(n^3)$ coefficients for one polynomial and $O(n^4)$ coefficients for the whole public key. As the key size of HFE is already rather big, such a big public key has almost certainly an impractical key size. Hence it is not recommended to have a private polynomial P' with the form $q^a + q^b + q^c$ for its powers.

4.4 HFE with More than One Branch

As we saw in Section 2.5, the most expensive step in HFE is root finding for signature generation or decryption. The running time of the corresponding algorithm grows with either the degree d of the private polynomial P or with the degree n of the extension field $\mathbb{E}$. In this section, we concentrate on the parameter n. One possible modification is HFE with two different branches as this reduces the degree n of the extension field $\mathbb{E}$. The overall structure is shown in Figure 4.1. Here the degree of the extension field is

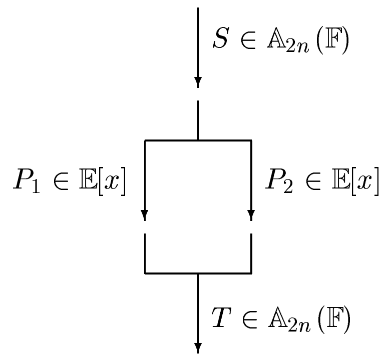

Figure 4.1: HFE with two branches P_1, P_2

n while it is possible to encrypt messages from $\mathbb{F}^{2n}$. As root finding can be done independently for both private polynomials P_1 and P_2, we expect a speed up of about one quarter in software (2 times $\frac{1}{8}$, assuming that the root finding algorithm is cubic in n, see Section 2.5) and $\frac{1}{8}$ in hardware.

Remark: A natural generalisation are more than two branches. For a branches, we expect a speed up of $aa^{-3} = a^{-2}$ in software and a^{-3} in hardware.

However, in [Bi00] Biham attacks an algorithm quite similar to HFE (from [PaGo97a]) which has many branches and the private polynomials are no bijections. The key point is that he exploits collisions, i.e., two different input values $x \neq x'$ with the same output value y for one branch. As his attack does not make any assumptions about the structure of the polynomial, it can easily be adapted for HFE. As a result, the extended and modified version of [PaGo97a] suggests to have at least 128 input variables while [Pa96] suggest $n \geq 64$ to be secure against this kind of attack. In turn, for a total input size of 128 bits and $\mathbb{F} = \mathrm{GF}(2)$, we can have at most two branches to avoid the Biham attack. According to [Pa96], using branches does not provide a significant speed up as the number of branches must be rather small. However, if the input size is larger (e.g., 256 bits), we think it can be worthwhile. As an overall result, having more than two branches will lead to a speed up for the rather expensive private key operation of root finding on costs of security. So this modification must be handled carefully and with respect to the intended application domain.

4.5 HFE-: Hiding Public Equations

While the first two changes discussed above tried to modify the private key of HFE, we now concentrate on the public key. Especially for signature schemes, an obvious change is to keep $1 < l < n$ polynomials $p_1, \ldots, p_l$ of the public key secret. As we discussed in Section 2.4, this is a necessity if the private polynomial P is not a surjection. However, even if the private polynomial P is a bijection, this might be a good idea as it keeps parts of the structure of the private key secret.

As for a signature scheme, keeping some polynomials $p_1, \ldots, p_l$ secret, is also expected to enhance the overall security of an encryption scheme. However, the number of equations removed can not be too high in this case. The reason for this restriction: keeping one equation p_1 secret effectively means to take $\log_2 q$ bits of information out of the encrypted message m' := HFE(m). To restore these missing $\log_2 q$ bits, it is necessary to try all q possibilities for $m'_1, \ldots, m'_q$ until the correct one is found. As the equation $P(x) = y$ has up to d solutions (see Section 2.2), we need to transmit some redundancy r anyway. However, to additionally compensate the loss of $\log_2 q$ bit of information, we will need to transmit more redundancy r. In addition, we need to solve up to q times the equation $P(x) = y$ for different values of y. As this operation is the most time consuming operation in HFE, this significantly slows down the decryption process. In general, by keeping l equations secret, we loose $\log_2 q^l$ bits of information and have to try up to q^l different possible encrypted messages $m'_1, \ldots, m'_{q^l}$, so we expect decryption to be $O(q^l)$ times slower.

However, in terms of an attack, this modification is very difficult to overcome. To see why this modification is so powerful, we will inspect one of the Patarin attacks from Section 3.5, namely the affine multiple attack. For this attack, we have to express x, y in terms of $\mathbb{F}$. For y being at most linear, we obtain the following equations over $\mathbb{F}$:

$$E_i : \sum_{1\leq j,k\leq n} \alpha_{i,j,k} x_j y_k + \sum_{j=1}^{n} \beta_{i,j} x_i + \sum_{j=1}^{n} \gamma_{i,j} + \delta_i = 0 \text{ with } 1 \leq i \leq n$$

Keeping one y_i secret, leads to several problems: first of all, it is no longer possible to compute the pair $(x, y) \in \mathbb{F}^n \times \mathbb{F}^n$, as one element of $y \in \mathbb{F}^n$ is missing. To compensate this loss, one possibility is to try all possible q settings $y_i = 0, \ldots, y_i = q-1$ for this element of y. As this must be done for all pairs (x, y), the number of modified pairs to try is q^β where $\beta \in \mathbb{N}$ is the number of pairs. As $\beta \gg n$, this attack is no longer more efficient than a brute force attack in the whole message space.

In the case of the private polynomial P being a bijection and $\mathbb{F} = \mathrm{GF}(2)$, it is possible to exploit these two facts and to counter HFE-. For any pair (x, y) and (x, y) with $y = y'$ (without the missing bit) and $x \neq x'$, we know that the two missing bits y_i and y'_i have the relation $y_i \neq y'_i$ as the private polynomial P is a bijection. [PGC98] is able to transfer this relation into an attack against HFE with P being a bijection. This attack is further generalised for more than one bit y_i kept secret. In general, i.e., when P is not a bijection, this attack does no longer work. However, for a sufficiently large number l of polynomials removed, even bijections are secure. For the rather weak Matsumoto-Imai polynomial (see Section 3.5), [PGC98] shows that $l > 10$ is secure when mixing the "+" and the "-" modification. When using the "-" modification alone, the same paper suggests $q^l > 2^{64}$ to be secure against all known attacks. In fact, the attack presented in [PGC98] has complexity $O(q^l)$, and for sufficiently large q, the number of equations removed can be much smaller. So using modifications HFE can be used with a bijective polynomial as private polynomial P, and is still expected to be secure. For an encryption scheme, the number of equations to be removed seems to be too high. So in this case, we need the private polynomial P to have all possible coefficients and also a rather high degree d. But even in this case, removing a small number $l \in \mathbb{N}$ equations seems to enhance the overall security of HFE significantly [Co01].

4.6 HFE+: Adding Public Equations

Rather than hiding equations from the public key, we can also add some randomly generated equations over $\mathbb{F}$ to the public key. The idea is to disturb the attacker by adding a kind of "noise" to the public key. It is possible to add these l new equations $q_1, \ldots, q_l$ to the already existing equations $p_1, \ldots, p_n$ using a new secret affine transformation $U \in \mathbb{A}_{n+l}(\mathbb{F})$, as suggested in [Pa96, p. 28]. After this step, we obtain quadratic polynomials $p'_1, \ldots, p'_{n+l}$ over $\mathbb{F}$. However, as $T \in \mathbb{A}_n(\mathbb{F})$ is an affine transformation, it

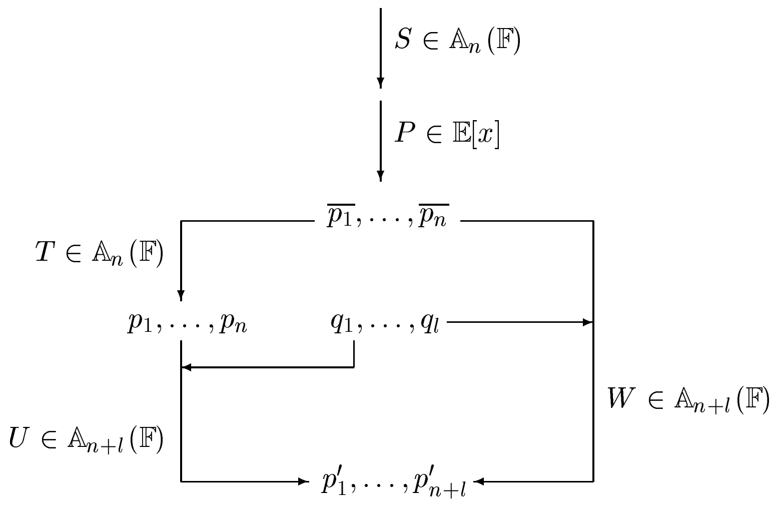

Figure 4.2: HFE+ with three (left) and two (right) affine transformations

will effect the polynomials $p_1, \ldots, p_n$ in an affine way. We show that it is not necessary to mix $p_1, \ldots, p_n$ and $q_1, \ldots, q_l$ using a new affine transformation U but that it is sufficient to mix them with an affine transformation $W \in \mathbb{A}_{n+l}(\mathbb{F})$ and to replace both affine transformations T and U by W. Figure 4.2 shows the key generation algorithm and gives a graphical representation of both ideas. Here $p'_1, \ldots, p'_{n+l}$ denotes the overall result, $\overline{p_1}, \ldots, \overline{p_n}$ are the polynomials after applying the private polynomial P, and $p_1, \ldots, p_n$ are the intermediate result after applying T.

To show that these two different ways of incorporating the random polynomials $q_1, \ldots, q_l$ are equivalent, we will study how the two affine transformations T and U affect the different polynomial vectors involved. Before we start, we express the affine transformation U as one matrix $M_U \in \mathbb{F}^{n+l \times n+l}$ and a vector $v_u \in \mathbb{F}^{n+l}$. The affine transformation T will be expressed non-standard. Rather than having one matrix $M_T \in \mathbb{F}^{n\times n}$ and one vector $v_t \in \mathbb{F}^n$, we will use a matrix $\overline{M_T} \in \mathbb{F}^{(n+l)\times(n+l)}$ and one vector $\overline{v_t} \in \mathbb{F}^{n+l}$. Using the coefficients $(m_t)_{i,j}$ of matrix M_T and the coefficients $(v_t)_i$ of vector v_t, they are coefficient-wise defined as follows:

$$(\overline{m_t})_{i,j} := \begin{cases} (m_t)_{i,j} & \text{, for } i,j \le n \\ 1 & \text{, for } i,j > n \text{ and } i = j \\ 0 & \text{, otherwise} \end{cases}$$

$$(\overline{v_t})_i := \begin{cases} (v_t)_i & \text{, for } i \le n \\ 0 & \text{, otherwise} \end{cases}$$

So in terms of matrix multiplication and vector addition, we can express the last two steps of HFE+ as

$$M_U \left[\overline{M_T} \begin{pmatrix} \overline{p_1} \\ \vdots \\ \overline{p_n} \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \overline{v_t} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ q_1 \\ \vdots \\ q_l \end{pmatrix} \right] + v_u = M_U \left[\overline{M_T} \begin{pmatrix} \overline{p_1} \\ \vdots \\ \overline{p_n} \\ q_1 \\ \vdots \\ q_l \end{pmatrix} + \overline{v_t} \right] + v_u$$

$$= (M_U \overline{M_T}) \begin{pmatrix} \overline{p_1} \\ \vdots \\ \overline{p_n} \\ q_1 \\ \vdots \\ q_l \end{pmatrix} + (M_U \overline{v_t} + v_u) = M_W \begin{pmatrix} \overline{p_1} \\ \vdots \\ \overline{p_n} \\ q_1 \\ \vdots \\ q_l \end{pmatrix} + v_w$$

for some $M_W \in \mathbb{F}^{(n+l)\times(n+l)}, v_w \in \mathbb{F}^{n+l}$. Moreover, as both matrices $\overline{M_T}$ and M_U are invertible, $M_W = M_U \overline{M_T}$ is also invertible. As all matrices are invertible, we can always compute one affine transformation for any two other given transformations. So from a cryptographic point of view, we can apply W directly to $(\overline{p_1}, \ldots, \overline{p_n}, q_1, \ldots, q_n)^t$ rather than working with the two transformations T, U. We show this with a counting argument: keep the transformation T fixed, and choose transformation U at random. As $\overline{M_T}$ is invertible, two different transformations U, U' will yield two different transformations W, W', so their number is the same. In terms of probability, each transformation W has the same probability to appear for T fixed and U chosen at random. This is also true when we allow different values for transformation T: for each T, there is one (and exactly one) U which yields a specific W. So the probability for a specific W to appear does not change by allowing T to have different values. So rather than choosing T and U at random and then compute W, we can choose W at random without changing the probability for any specific $W \in \mathbb{A}_{n+l}(\mathbb{F})$ to appear.

In terms of speed, this simplified HFE+ is not expected to be much faster then the original one. Applying (or inverting) an affine transformation is a rather cheap operation - at least compared with solving polynomial equations (see Section 2.5). However, we save one affine transformation from $\mathbb{A}_n(\mathbb{F})$ in the private key, so we do not need to store $n^2 + n$ elements of $\mathbb{F}$, which reduced the private key size by a little less than one third (see Section 2.7).

For an encryption scheme, HFE+ is certainly an option as it does not hide information but gives extra redundancy to the message. For a signature scheme, it provides a problem as these extra equations must be satisfied without having a trapdoor. From a practical point of view, only $\frac{1}{q^l}$ of all solutions for a given signature $h(m)$ will satisfy these extra equations $q_1, \ldots, q_l$. So l should not be too large in a signature scheme. In the light of the latest attacks (see Section 3), l should not be to large for an encryption algorithm either as an slightly over-defined system seems to be far easier to solve than a system with less equations than variables. As a result from our current knowledge, HFE+ alone does not enhance the security of HFE but seems to impose a threat to its security and must therefore be handled with care.

4.7 HFEv: Adding Vinegar Variables

While HFE- and HFE+ change the public key, adding vinegar variables, i.e., HFEv, changes the structure of the private polynomial P. Instead of using one private polynomial P, this modification allows q^v many private polynomials $P_1, \ldots, P_{q^v}$ where $v \in \mathbb{N}$ denotes the number of vinegar variables $z_1, \ldots, z_v$. As the private key should still be expressible in terms of at most quadratic polynomials $p_1, \ldots, p_n$, there is a restriction on the way these q^v many private polynomials are obtained. In essence, the quadratic coefficients (i.e., coefficients with a power of the form $q^a + q^b$ for some $a, b \in \mathbb{N}$) have to be the same for all these polynomials, while the linear coefficients depend on these vinegar variables $z_1, \ldots, z_v$ in an affine way, and the constant term depends on them in an at most quadratic way. In symbols:

$$\begin{aligned} P_{(z_1,\ldots,z_v)} \quad := \quad & \sum_{\substack{0\le i,j\le d \\ q^i+q^j\le d}} \alpha_{i,j} x^{q^i+q^j} + \sum_{\substack{0\le k\le d \\ q^k\le d}} \beta_k(z_1,\ldots,z_v) x^{q^k} + \gamma(z_1,\ldots,z_v) \\ & \text{for } \alpha_{i,j}\in\mathbb{E}, \\ & \beta_k(z_1,\ldots,z_v) \text{ are affine in } (z_1,\ldots,z_v)\text{, and} \\ & \gamma(z_1,\ldots,z_v) \text{ is at most quadratic in } (z_1,\ldots,z_v) \end{aligned}$$

For a signature scheme, HFEv can be implemented very easy. The vinegar variables $(z_1,\ldots,z_v) \in_R \mathbb{F}^v$ are initialised with random values. After this step, there is only one private polynomial P, so the rest of the algorithm keeps unchanged. For an encryption scheme, it is not so easy to introduce this "v" modification as a priori any of the q^v possible settings for the vinegar variables is equally alike. As in HFE-, it is necessary to check up to q^v different equations $P(x) = y$ - but in this case, not the value of y but the polynomial itself changes. So for a signature scheme, it is possible to have many vinegar variables, while in an encryption scheme, their number must be small.

4.8 HFEf: Fixing Input Variables

As for most of the previous modifications, HFEf ("fixing") tries to disguise the structure of the private key by removing information from the public key. In this case, the information is removed by fixing $1 \le f \le n$ input variables $x_1,\ldots,x_f$ with random values from $\mathbb{F}^f$. As a result, the public key becomes shorter, as it no longer consists of n polynomials in n variables, but of n polynomials in $n-f$ variables. This fixing results in a partly evaluation of the public key. For example, terms $a_i x_i$ with $1 \le i \le f, a_i \in \mathbb{F}$ will be added to the constant term a_0, while terms $a_{i,j}x_ix_j$ with $1 \le i \le f < j \le n, a_{i,j} \in \mathbb{F}$ will be added to the corresponding linear term $a_j x_j$ for $a_j \in \mathbb{F}$.

For an encryption scheme, this modification reduces the number of bits which can be transmitted by $\log_2 q^f$. However, it does not result in slowing down the decryption process. In a signature scheme, the situation is more serious. Assuming that each input x_i has a specific value from $\mathbb{F}$ with probability q, only $\frac{1}{q^f}$ of all possible solutions for the problem $y = \mathrm{HFE}(x)$ for given y will have the correct values for $x_1,\ldots,x_f$. As a result, the process of finding a signature is expected to be slowed down by $O(q^f)$, so for a signature scheme, f must be rather small. In the light of the latest attacks (see Section 3.6), it seems to be a good idea to keep f small, even in the case of an encryption scheme, to have about the same number of equations as variables.

For HFEf in general, it seems to be obvious that it is a good idea not to choose the first $x_1, \ldots, x_f$ variables, but f randomly chosen variables from the set of all variables $\{x_1, \ldots, x_n\}$ and to fix them with a random choice $(r_1, \ldots, r_f) \in_R \mathbb{F}^f$. In fact, [Co01a, Sec. 12.1] seems to suggest exactly this when writing "[HFEf] consists of fixing some f input variables of the public key". However, we show that this is not the case. For this purpose, we introduce a permutation $\pi \in S_n$, and evaluate the variables $x_{\pi(1)}, \ldots, x_{\pi(f)}$ with elements from $\mathbb{F}$ while $x_{\pi(f+1)}, \ldots, x_{\pi(n)}$ become the new variables $x'_1, \ldots, x'_{n-f}$. Let $M_R \in \mathbb{F}^{n \times n}$ be the corresponding permutation matrix to π. Moreover, let the affine transformation S be expressible as the matrix $M_S \in \mathbb{F}^{n \times n}$ and the vector $v_s \in \mathbb{F}^n$. We can now express the first step in HFEf as

$$\begin{aligned} S(\pi(x_1, \ldots, x_n)) &= M_S(M_R(x_1, \ldots, x_n)^t) + v_s \\ &= (M_S M_R)(x_1, \ldots, x_n)^t + v_s \\ &= M'_S(x_1, \ldots, x_n)^t + v_s \end{aligned}$$

for some $M'_S \in \mathbb{F}^{n \times n}$. As both M_S and M_R are invertible matrices, so is M'_S. In fact, for any choice of an affine transformation S and a permutation π, there is exactly one affine transformation $S' \in \mathbb{A}_n(\mathbb{F})$, which has exactly the same effect as $S(\pi(\cdot))$. So from a cryptographic point of view, π has no effect, as both S and S' are unknown affine transformations without any internal structure. As both have the same size, namely consists of $n^2 + n$ a priori unknown elements from $\mathbb{F}$, it is equally hard to find either S or S'. In the first case, we also have to reveal the secret permutation π. So for cryptographical analysis, it seems to be preferable to look for S' rather than both S and π. In fact, for a crypto-graphical analysis we can simply assume that permutation π does not exist and concentrate on finding a suitable affine transformation $S' \in \mathbb{A}_n(\mathbb{F})$. Using the equation above we know that such a transformation must exist.

Remark: It seems to be obvious that the affine transformation S' is unique for a given public key k. In fact, S' is unique when both S and π are fixed. However, it is possible that there are two pairs of affine transformations S, T, which lead to the same public key for a given private polynomial P. As we cannot rule this out a priori, we cannot assume that S is unique for a given public key and therefore can not assume that affine transformation S' is uniquely determinate for a given public key k.

4.9 HFEm: Imitating Vinegar Variables for Encryption

As already outlined in Section 4.7, vinegar variables are no problem for a signature scheme. But for an encryption scheme, they provide a serious challenge. In essence, the number of vinegar variables must be rather small as the decryption process is slowed down by $O(q^v)$ where v denotes the number of vinegar variables.

So the aim of this section is to study the effect of introducing so called "masking variables" at a later stage of HFE, namely after the private polynomial P is applied. This version is called HFEm. The general idea is outlined in Figure 4.3. The author is not aware that this modification has been discussed elsewhere.

Here $q_1, \ldots, q_m$ ("masking polynomials") are randomly generated polynomials in $n+m$ input variables while $\overline{p_1}, \ldots, \overline{p_n}$ ("trapdoor polynomials") are an intermediate result after the affine transformation S and the private polynomial P. Both are mixed using the affine transformation $T \in \mathbb{A}_{m+n}(\mathbb{F})$ and yield the result polynomials $p_1, \ldots, p_{m+n}$. The trapdoor polynomials $\overline{p_1}, \ldots, \overline{p_n}$ have $n+m$ input variables, too. All p_i, $\overline{p_j}$, and q_i are over $\mathbb{F}$. The fact that m randomly generated polynomials are added rather than $m' \in \mathbb{N}$ with $m \neq m'$ is only due to the sake of simplicity. Note: the extension field $\mathbb{E}$ has dimension n, rather than $n+m$.

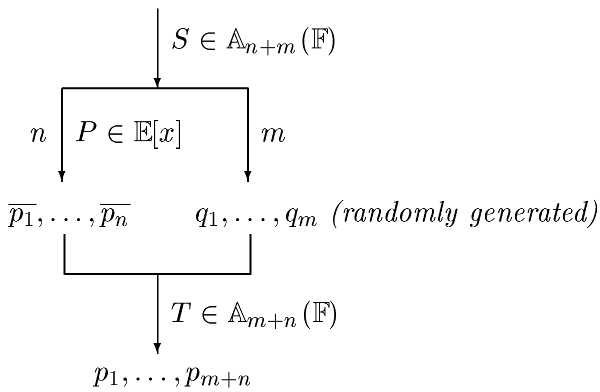

Figure 4.3: HFE with masking variables

The following section describes how to use HFEm in an encryption/decryption scheme. For encryption, we use the polynomials $p_1, \ldots, p_{n+m}$, and also add redundancy r, as described in Section 2.3.3. For decryption, it is still possible to use the trapdoor (S, P, T), however, we also have to incorporate $q_1, \ldots, q_m$ into this process. For the following algorithm, we assume that y is given and we want to obtain the original message m:

1. Reverse the affine transformation T and obtain $y' \in \mathbb{F}^{n+m}$.

2. Transfer $y'_1, \ldots, y'_n$ to $\overline{y} \in \mathbb{E}$, using the bijection discussed in Section 2.1.

3. Solve the polynomial equation $P(\overline{x}) = \overline{y}$ and obtain $d' \leq d$ different results $R := \{\overline{x_1}, \ldots, \overline{x_{d'}}\}$.

4. For each result from R, there are m coefficients from $\mathbb{F}$ undefined, so we obtain $|R|q^m$ possible solutions in total. Call the set of these solutions R'.

5. Reverse each solution from R', using the affine transformation S. Using the redundancy r, or the polynomials $q_1, \ldots, q_m$, we are able to pick the correct message m.

As we still get q^m solutions, this modification does not seem to solve our original problem as it still slows down decryption in terms of $O(q^m)$. Even worst, signature generation is also slowed down $O(q^m)$, using a similar argument as in the previous section. However, for HFEv, it was necessary to repeat the rather expensive step of root finding up to q^v times. For HFEm, it is sufficient to find all possible roots once, and then iterate over all possible q^m solutions for every given solution from the set R. This is obviously much faster than repeated root finding.

So HFEm and HFE+ are very similar in the way they add randomly generated polynomials. However, in HFEm we are able to add variables at the same time, so the system of equations is no longer over-defined. This modification is therefore a possibility to deal with the threat from FXL-like algorithms and add randomness to the public key at the same time.

4.10 HFEm', HFEz: Zero Added Equations

To generalise HFEm, it seems natural to allow the number of variables and the number of equations to be distinct, i.e., to add m variables in m' randomly generated equations. As our overall goal for HFEm was to have more variables than equations (like in HFEv), we get the largest effect for $m' = 0$. That is, we add zero new equations. This seems to be strange as there does not seem to be any room for the new variables. However, the input for the private polynomial P still depends on all $m + n$ coefficients from $\mathbb{F}$. So all $m+n$ variables are needed to get the correct value for any of these n input coefficients. As for HFEm, the author is not aware that this modification has been discussed elsewhere.

We will illustrate HFEz by computing the public key of a toy example using base transformation (see Section 2.3.2). The parameters of our toy example are: $n = 2, m = 1$, $\mathbb{F} = \mathrm{GF}(2)$, $i(t) = t^2 + 1$, $\mathbb{E} = \mathbb{F}[t]/i(t)$. Moreover, let

$$\begin{aligned}
S(x_1, x_2, x_3) &:= \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \\
T(x_1, x_2) &:= \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\
P(x) &:= x^3 + 1
\end{aligned}$$

By expressing S in terms of affine polynomials p_1, p_2, p_3, we obtain:

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

However, the next step is different to base transformation as seen in Section 2.3.2. As we have $m = 1$, we have to discard one polynomial. Without loss of generality, we reject p_3. So S is expressed as

$$\begin{aligned}
\mathfrak{P}[t] &= p_2 t + p_1 \\
&= (x_2 + x_3)t + (x_1 + x_2)
\end{aligned}$$

to mimic operations in $\mathbb{E}$. To obtain the private polynomial P over $\mathbb{F}^{n+m}$, we compute $x^3 + 1$ in terms of polynomial computations and denote the result $\mathfrak{Q}[t]$. In the following computation, $\equiv$ denotes reduction by the irreducible polynomial $i(t)$.

$$
\begin{aligned}
\mathfrak{Q}[t] &= (\mathfrak{P}[t])^3 + 1 \\
&= (\mathfrak{P}[t]^2)\mathfrak{P}[t] + 1 \\
&= [(x_2 + x_3)t + (x_1 + x_2)]^2[(x_2 + x_3)t + (x_1 + x_2)] + 1 \\
&= [(x_2 + x_3)t^2 + (x_1 + x_2)][(x_2 + x_3)t + (x_1 + x_2)] + 1 \\
&\equiv [x_1 + x_3][(x_2 + x_3)t + (x_1 + x_2)] + 1 \\
&= (x_3 + x_1x_2 + x_1x_3x_2x_3)t + (x_1 + x_1x_2 + x_1x_3x_2x_3 + 1)
\end{aligned}
$$

Before we can apply an affine transformation, we have to transfer the result to a two dimensional vector, denoted $\mathfrak{Q}(x_1, x_2, x_3)$.

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

We now apply affine transformation T to the vector $\mathfrak{Q}(x_1, x_2, x_3)$, denoting the result $\mathfrak{R}(x_1, x_2, x_3)$:

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

The result $\mathfrak{R}(x_1, x_2, x_3)$ clearly depends on x_1, x_2, x_3, i.e., on all 3 input variables.

To revert HFEz, i.e., for a encryption/decryption scheme, we use the same techniques as for HFEm (see Section 4.9). The overall scheme can be seen in Figure 4.4. The picture is very similar to HFEm. However, the m masking variables, introduced in the affine transformation S, and mixed with the other n input variables, do not lead to m randomly generated equations. They are instead discarded. So $m + n$ input variables lead to n equations in the public key. For a signature scheme, m can be much bigger than for an encryption scheme. However, even in an encryption scheme, m can be larger than the parameter v in an HFEv ("vinegar") scheme, so this modification can be worthwhile for an encryption scheme.

We call the more general case $1 \leq m' < m$ an HFEm' scheme. Here we add m' random equations and m new variables. The advantage over

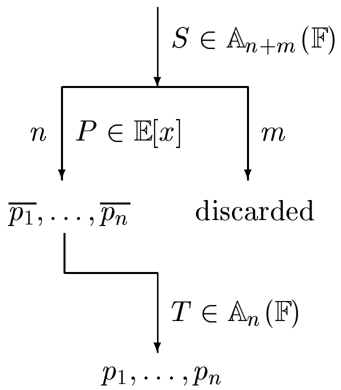

Figure 4.4: HFE with masking variables but with zero added equations

an HFEz scheme is to add some "randomness" to the trapdoor while the advantage over an HFEm scheme is the fact that we add more variables (like in an HFEv scheme).

4.11 Modifications Revisited

As we saw in this section, HFE is a very flexible scheme which allows many modifications. Thus HFE can be adapted to different application domains and also react on different attacks by slightly changing its overall structure. In the remainder of this section we go briefly through these different variations on HFE and determine how useful they are for an encryption or a signature scheme.

For a signature scheme, HFE- and HFEv are certainly very useful as they do not lead to any slow down for signature generation and enhance the overall security of HFE. In contrast, both HFE with more branches (see Section 4.4) and HFE using a subfield of $\mathbb{F}$ for the public key (see Section 4.2), seem to lead to a less secure scheme. On the other hand, both modifications change HFE in a very desirable way: the first leads to a speed up for signature generation while the second yields a smaller public key. So for certain parameter choices, both variations can lead to secure signature algorithms. In contrast, HFE+, HFEf, HFEm, and HFEz are no good choice for a signature scheme. For example, HFEf will slow down significantly signature finding, but has similar effects as HFE- in terms of the public key. Finally, using bijections as private polynomials seems to be a way to speed up the signature generation process. Combining this modification with HFE- can be used to construct both fast and secure public key signature (see Section 5.1).

For encryption, the situation seems to be worse. Both HFE- and HFEv will lead to a rather slow decryption process so neither too many equations can be removed (HFE-) nor too many variables added (HFEv). For HFEf and HFE+, the situation is better. In both cases, the decryption step takes the same amount of time, however, the public key has more equations than variables. As solving over-defined equations in many variables looks sub-exponential (see Section 3.6), it seems to be a good idea to have nearly the same number of equations as variables. A special choice for the private polynomial P, e.g., a bijection, is not secure. And combining this with HFE- does not yield results either, as we cannot remove too many equations from an encryption scheme. Having two or more branches for the private polynomial P or using a subfield of $\mathbb{F}$ seems to have no special effect for an encryption scheme, so the same caution is necessary as for a signature scheme. However, both HFEm and HFEz can be of particular use for an encryption scheme. They lead to more variables than basic HFE but do not lead to repeated calling of the expensive root finding step. So they can add far more variables than HFEv, and still lead to a reasonable decryption time.

Chapter 5

Applications

In the previous section, we were dealing with different modifications of HFE to make it either faster or to enhance its security. After this more theoretical approach, we show in this section, how these modifications can be used to obtain different versions of HFE for "real world" public key cryptography.

5.1 Flash / SFlash

We start with two NESSIE submissions, namely Flash and SFlash. NESSIE (New European Schemes for Signatures, Integrity, and Encryption) is a research project, funded by the European Union under the Information Societies Technology (IST) Programme, to standardise "cryptographic primitives" [NESSIE]. Apart from other areas (e.g., symmetric ciphers, cryptographic hash functions), NESSIE deals with public key signature algorithms. Flash and SFlash have been submitted by Courtois, Goubin, and Patarin [CGP00]. In this section, we describe both algorithms and their modifications during the NESSIE process. Where not stated otherwise, this section is based on [CGP00] and [CGP01]. As the final evaluation of the NESSIE proposals is not due before March 2003 [NESSIE], it is not clear yet which cryptographic primitives will be selected.

Both Flash and SFlash are a HFE- signature schemes which use a bijection as private polynomial. In contrast to HFE, the private polynomial P is no longer private, but publicly know. The reason for this change lies in the fact that both signature algorithms have so many public equations removed that this is not expected to be a threat for Flash's or SFlash's security. As we see in Figure 5.1, there is a markable difference in the public key size of Flash and SFlash. In fact, this difference is much higher than expected: as

Parameter	SFlash	Flash
$q = \lvert\mathbb{F}\rvert$	$128 = 2^7$	$256 = 2^8$
$n = \partial i(t)$	37	
l (equations removed)	11	
$P(x)$	$x^{128^{11}+1}$	$x^{256^{11}+1}$
Signature Length	259 bits	296 bits
Private Key Size	0.35 kb	2.75 kb
Public Key Size	2.2 kb	18kb

Figure 5.1: Parameters for the First Version of Flash and SFlash

Flash is based on the finite field $\mathbb{F} = \mathrm{GF}(2^8)$ and has a public key size of 18kb, SFlash is based on the finite field $\mathbb{F} = \mathrm{GF}(2^7)$. Therefore we would expect a public key size of 18kb$\frac{7}{8} \approx$ 16kb. The reason for this difference is due to a "trick" from Section 4.2, namely the restriction of the coefficients in both affine transformations S and T to a subfield of $\mathbb{F}$. In this case, coefficients for these two transformations come from $\widetilde{\mathbb{F}} = \mathrm{GF}(2)$ rather than $\mathbb{F} = \mathrm{GF}(2^7)$. And in fact, 18kb$\frac{1}{8} \approx$ 2.2kb, so the public key size (and accordingly the private key size) are within the expected range.

Parameter	SFlash
$q = \lvert\mathbb{F}\rvert$	$128 = 2^7$
$n = \partial i(t)$	37
l (equations removed)	11
$P(x)$	$x^{128^{11}+1}$
Signature Length	259 bits
Private Key Size	2.45 kb
Public Key Size	15.4 kb

Figure 5.2: Parameters for the Second Version of SFlash

However, due to the attack from [GSB01], the submission of SFlash was changed. Moreover, Martinet suggested in [Ma01a] to concentrate on either Flash and SFlash but not both algorithms as they are very similar. Due to the shorter signature length, the higher speed and also the shorter public key, she suggested to concentrate on SFlash. According to [Pr+02], NESSIE followed this suggestion, so Flash is no longer in the standardisation process of NESSIE.

In fact, to avoid the attack from [GSB01], SFlash does no longer use a subfield $\widetilde{\mathbb{F}}$ of the finite field $\mathbb{F}$ but chooses the coefficients for the two affine transformations S and T from the whole field $\mathbb{F} = \mathrm{GF}(128)$. The overall parameters for SFlash, version 2, are summarised in Figure 5.2. As [Ma01] points out, SFlash (she was dealing with the first version of SFlash) has very well chosen parameters to avoid all known attacks against it. In fact, the change from Version 1 of SFlash to Version 2 of SFlash as outlined above, follows exactly this path. So from our current knowledge, SFlash is secure against all known attacks. On the other hand, it is very new and the last attack dates back only one year. So it may be too early to use SFlash at presence.

5.2 Quartz

In contrast to Flash and SFlash, Quartz was not only designed to withstand all known attacks but also to have good chances to withstand coming attacks as well. So in contrast to Flash/SFlash, all parameters in Quartz are chosen very conservatively, which results in a rather long signature time, namely 10s on average on a Pentium II 500MHz [CGP01]. When not stated otherwise, this section is based on [CGP01] and describes the second, reversed version of Quartz. In contrast to SFlash, the changes made from the first version to the second version of Quartz are not due to security problems. On the one hand, they were made to speed up the whole algorithm without jeopardising its security. On the other hand, they allow a security proof for Quartz.

Parameter	Quartz
$q = \|\mathbb{F}\|$	2
$n = \partial i(t)$	103
transformation S	$\mathbb{F}^{107} \to \mathbb{F}^{107}$
transformation T	$\mathbb{F}^{103} \to \mathbb{F}^{103}$
l (equations removed)	3
v (vinegar variables)	4
Signature Length	128 bits
Private Key Size	3 kb
Public Key Size	71 kb

Figure 5.3: Parameter for Quartz

As we see in Figure 5.3, the signature length (128 bits) is 28 bits larger than expected: as the extension field $\mathbb{E}$ has dimension 103 while 3 equations are removed, we would expect a signature length of 100 bits. The reason for this difference lies in the fact that Quartz uses a so-called "Feistel-Patarin-Network" to compute the signature. Within this network, the HFE algorithm is called four times to compute a signature, i.e., this involves solving the underlying HFE problem four times.

To deal with the different security features of Quartz, we have to look at them and try to deduce if the enhance or jeopardise the security of Quartz. First of all, the private polynomial P has full coefficients, i.e., it has non-trivial coefficients from $\mathbb{E}$ and also all possible coefficients, i.e., every power which has Hamming weight two or lower. Together with the vinegar variables (denoted $z_1, \ldots, z_4$), the private polynomial P of Quartz can be expressed as:

$$\begin{aligned} P_{(z_1,\ldots,z_4)} \quad := \quad & \sum_{\substack{0\leq i,j\leq 7 \\ q^i+q^j\leq 129}} \alpha_{i,j} x^{q^i+q^j} + \sum_{\substack{0\leq k\leq 7 \\ q^k\leq 129}} \beta_k(z_1,\ldots,z_v) x^{q^k} + \gamma(z_1,\ldots,z_v) \\ & \text{for } \alpha_{i,j} \in \mathbb{E}, \\ & \beta_k(z_1,\ldots,z_v) \text{ are affine in } (z_1,\ldots,z_v) \text{, and} \\ & \gamma(z_1,\ldots,z_v) \text{ is at most quadratic in } (z_1,\ldots,z_v) \end{aligned}$$

As the polynomial has all coefficients and its rather high degree, Quartz withstood all known attacks up to a security level of 2^{80} - this level is requested for signature algorithms in NESSIE. This is also true if there is no "v" modification. In fact, the degree is very high as [HFE] suggests that d=$25-33$ is high enough. In addition, Quartz is a HFEv rather than a "basic" HFE scheme. This modification is expected to enhance the security of Quartz further. Moreover, Quartz is also a HFE- scheme with 3 equations kept secret. Although it is possible to attack some HFE- schemes with l equations removed in $O(q^l)$ (see Section 4.5), these schemes must be bijections to make this attack working. As Quartz uses a very general polynomial P and also the "v" modification, these attacks do not apply to Quartz. So removing only three equations from the public key seems to be sufficient for Quartz and is expected to enhance its overall security. We call the parts of Quartz discussed above the "HFE"-step, i.e., HFE$(x) := T(P(S(x)))$ and its inverse HFE$^{-1}(y) := S^{-1}(P^{-1}(T^{-1}(y)))$.

Although the HFE-step itself looks quite secure, there is an obvious attack using the birthday paradox: computing 2^{50} different versions of the message m and also apply the public key to 2^{50} different values for $x_1, \ldots, x_n$

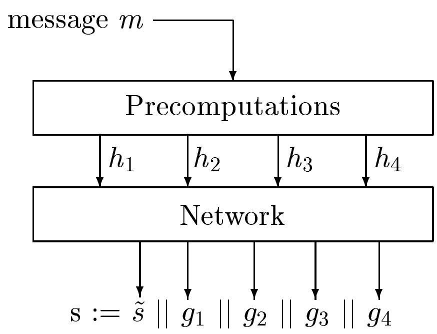

Figure 5.4: Overall Structure of Quartz for Signature Generation

we expect to obtain a valid signature for one version of the message m as the HFE step alone has only a 100 bit signature. This is far less then the security level of 2^{80} required in NESSIE. To overcome this problem, Quartz combines four invocations of the HFE-step in a so-called "Feistel-Patarin network". The key idea of this network is not to store four times a whole signature (i.e., a signature of 400 bits in total) but to save only one, namely the last signature completely. In addition, is stores 7 bit for each of the 4 signatures computed. The reason for this lies in the fact that the HFE step of Quartz has a 100 bit input but only a 107 bit output. These additional 7 bit compensate this "loss". The overall structure of this network is shown in Figure 5.4. As we see there, signature generation with Quartz requires a precomputation step (see Figure 5.5) before applying the network itself (see Figure 5.6). The key idea of the precomputation step is to use three calls of a 160 bit hash function (SHA-1 in Quartz) to "expand" a 160 bit hash (denoted m_0 in Figure 5.5) to four 100 bit values h_1, h_2, h_3 and h_4. During this process, the original hash m_0 is concatenated (operator $\cdot||\cdot$) with the 8 bit values $0x00$, $0x01$ and $0x02$ (C notation for the numbers 0, 1 and 2) to obtain three 168 bit values. Each of them is hashed individually using a 160 bit hash function and then concatenated. The resulting 480 bit number is truncated to 400 bits and yields four 100 bit strings. If Quartz used a hash function with a 512 bit output rather than 160 bit, the precomputation step would be obsolete. Such functions are currently under consideration in the NESSIE project (algorithm "Whirlpool") and in NIST [NIST01]. In both cases, the standardisation process is not finished at the moment. But for the security level of 2^{80}, it is sufficient to use a 160 bit hash function and to expand its output to 400 bits as done in the precomputation step of Quartz.

We will now concentrate on the "Feistel-Patarin network" itself as out-

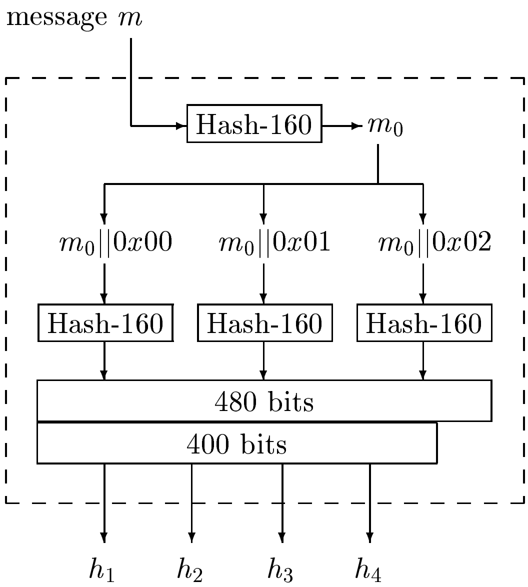

Figure 5.5: Precomputation in Quartz

lined in Figure 5.6. It uses the output of the precomputation step as input. We describe the first step of the network. After initialising the counter i with 0, it "xors" h_0 with 0 to obtain the intermediate value y. This is certainly obsolete as h_0 "xor" $0 = h_0$. However, during the run of the algorithm, h_1, h_2, h_3 are "xored" with the output of the previous step, so this "xor" operation is required for symmetry of the four steps. After this initialisation, the 100 bit value y is hashed together with a secret 80 bit parameterΔ to obtain the random variables r (3 bit) and the vinegar variables z (4 bit). Both are fed into the HFE step to obtain a valid signature of 107 bit. According to [CGP01, Sec. 5.3], the probability to obtain a valid signature at the first try is $\approx 60\%$. If there is no valid signature, the hash of $y \mid\mid \Delta$ is rehashed. This is repeated until a valid signature is obtained. The probability that there is no valid signature at all for a given message m is estimated to be $\leq 2^{-183}$ and hence negligible [CGP01, Sec. 5.3]. If a valid 107 bit signature is found, the least significant 100 bits of x are fed back into the network while the most significant 7 bits are stored as output g_1. The other three steps are similar but h_i is not "xored" with 0 but with the least significant 100 bits of x. In the final step, these 100 bits are not fed into the network but yield output $\tilde{s}$. In each step, it is possible that there is not only one, but up to $d = 129$ different solutions for the equation $x = HFE(y \mid\mid r)$. The Quartz-standard states that only one is chosen, namely the one with

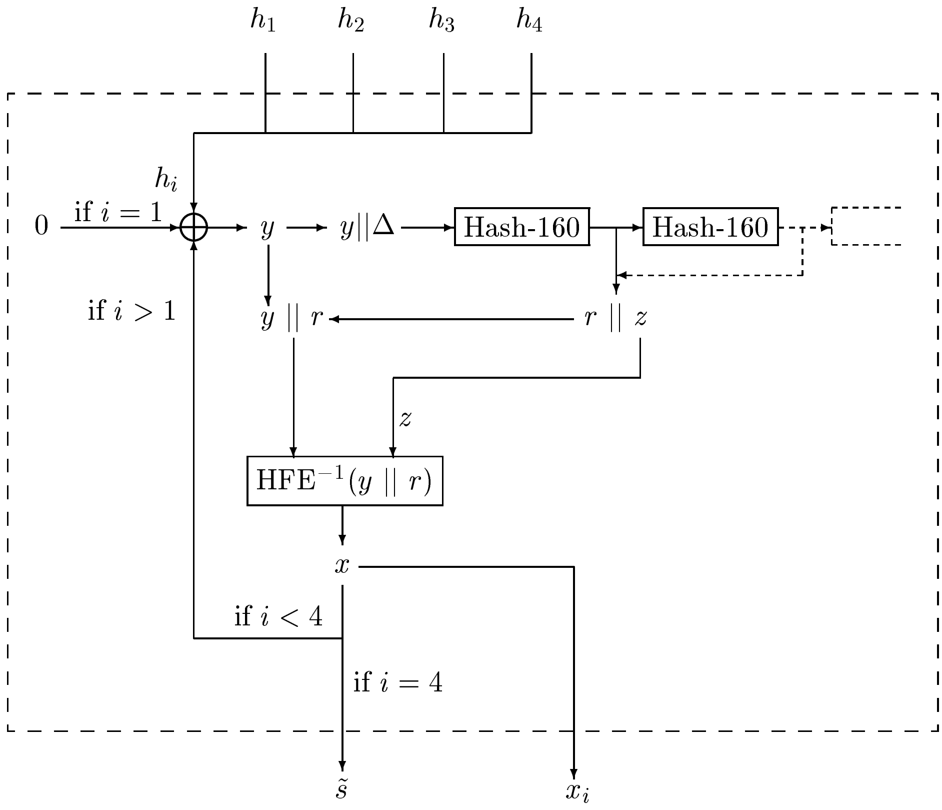

Figure 5.6: Central Structure of the Feistel-Patarin network for Quartz

the least hash value (bit-wise comparison without sign bit).

To verify the validity of a signature, this network is reversed. As the public key consists of 100 polynomials $p_1, \ldots, p_{100}$ in 107 input variables $x_1, \ldots, x_{107}$, the 7 bit values $g_1, \ldots, g_4$ are used to obtain 107 bits input for the public key during each run. In addition, as the four 100 bit values $h_1, \ldots, h_4$ are "xored" each time, a signature is only valid if the overall output of this scheme is 0. In this case the signature is accepted. According to [Pr+02], Quartz is still in the evaluation process of NESSIE.

The Feistel-Patarin network is certainly a rather complicated security feature. However, as each signature depends on a 400 bit input (which is obtained from a 160 bit hash value), it seems to be a rather strong signature system. The NESSIE process did not find any weakness in Quartz [Pr+02]. Moreover, as Quartz uses the 160 bit hash function as a kind of cryptographically secure random number generator, it is deterministic, so each massage has always the same signature (for the same private key K). In the original specification of HFE as a signature scheme it seemed to be necessary to use "real" randomness to obtain valid signatures. As real randomness often is a problem (e.g., in a stand alone server without user interaction), this modification makes it possible to use Quartz in more application domains.

However, the latest attack from [CDF02] estimates that Quartz has a too low degree but needs $d = 257$ to be secure. As this attack relies heavily on empirical evidence, it is an interesting question if Quartz will be selected in the NESSIE process. The answer is not expected before March 2003 [NESSIE].

5.3 Shared Key Generation

In this section, we present another way of using "Hidden Field Equations" for signature. In contrast to the previous section, we are now concerned that it is not possible to use a signature algorithm for encryption. This restriction may looks a little strange but has a legal reason: in some states (including the USA) there are strong restrictions on the export (sometimes the use of cryptography, e.g., France) of products which can be used for encryption [Sc96, Sec. 25.14-15]. For software which cannot be used for this purpose but for electronic signature, these restrictions often do not apply. One problem in this context is the so-called "centralised leakage" [Co01a, Sec. 2.3] problem. Here we are concerned about the possibility that the key of an algorithm A (e.g., HFE) can be used as key for another algorithm B (e.g., RSA). In general, there is no solution for this problem [Co01a, Sec. 2.2] as usually both algorithms A and B expect a bit-string as key and there is no way to avoid a bit-string from being reused. However, some encryption algorithms require a special form for the key (e.g., RSA, see Section 1.2). In this case, a user who wants to abuse the system needs to "embed" a special key (e.g., for RSA) into the key of a signature scheme (e.g., HFE). One obvious solution to prevent this kind of abuse is centralised key generation, e.g., to have one central authority which generates all keys for all users. However, this system has several disadvantages. Apart from the fact that this centralised key generation service usually needs a high computational power to generate all keys for all users it also knows the private keys of all users - at least during the key generation step. This is certainly a problem if the user does not fully trust this service.

In [Co01a] both problems are addressed by "shared key generation" for

HFE. Figure 5.7 shows how to generate a public/private key pair (k_x, K_X) in cooperation between the user Alice and the central service Trent (sketch). Both Alice and Trent generate affine transformations S, S', T and T' while

1. Alice generates a private key $K_A = (S, P, T)$ and computes the corresponding public key $k_a = (p_1, \ldots, p_n)$.
2. Alice transmits her public key k_a to Trent.
3. Trent generates two affine transformations S' and T' and applies both to the public key k_a.
4. Trent transmits the two transformations S' and T' to Alice.
5. Alice modifies her private key to $(S' \circ S, P, T \circ T')$.

Figure 5.7: Shared Key Generation between user Alice and centre Trent

only Alice generates the private polynomial P. Alice's public key k_a depends on all five parts. As the concatenation of two affine transformation is again an affine transformation (see Section 4.6), the size of Alice's public key k_a does not increase with shared key generation. In addition, Trent does not need to know Alice's trapdoor to modify her public key.

We now try to attack this algorithm: assume that Trent wants to forge Alice's signature (see Section 3). One possibility is to find the trapdoor for her public key k_a. Although he knows S' and T', the problem of finding (S, P, T) is as difficult as finding $(S' \circ S, P, T \circ T')$ for any other person. So Trent has no advantage for a structural attack. Another way of forging her signature is to find the solution for a quadratic system of equations. Again, Trent does not have any advantage over any other person to apply an inversion attack. In contrast, Alice has difficulties to embed information into her public key. Assume that the first 1024 bits of her public key (S, P, T) can also be used as an RSA key R. In this case, the affine transformations S' and T' destroy this key. In addition, Alice cannot be sure that the new key R' (i.e., the first 1024 bits of her public key) is a valid RSA key. Even if it were a valid key, Alice does not know the prime factorisation of R' and hence cannot use it to decipher messages. So shared key generation with HFE as outlined in Figure 5.7 can be used to avoid the centralised leakage without revealing the private key to a central authority. Although [Co01a] does not address this problem, shared key generation also improves the security of the key if either Trent or Alice do not use a cryptographically secure random number generator (CSPRNG). In this case, the other random number generator adds "real" randomness to the key. However, as Trent does not modify the private polynomial P, a weakness in Alice's random number generator is likely to lead to a weakness in this part of her private key.

The overall system presented in [Co01a, Def. 19.0.1], see Figure 5.8, also

1. Alice generates a private key $K_A = (S, P, T)$ with $S \in_R \mathbb{A}_{n+v}(\mathbb{F})$, $T \in_R \mathbb{A}_n(\mathbb{F})$, $P_{(z_1,\ldots,z_v)} \in_R \mathbb{E}[x]$ and $P_{z_1,\ldots,z_v}$ an HFEv polynomial. Then she computes the corresponding public key $k_a = (p_1, \ldots, p_{n-l})$.

2. Alice transmits her public key k_a to Trent.

3. Trent generates l' random quadratic equations $(q_1, \ldots, q_{l'})$ in $n + v$ variables and mixes them with Alice's public key k_a. Call the result k'_a.

4. Trent generates two affine transformations $S' \in_R \mathbb{A}_{n+v}(\mathbb{F})$ and $T' \in_R \mathbb{A}_{n+v}(\mathbb{F})$ and applies both to the public key k'_a. Moreover, he fixes f variables in k'_a. Denote the result of this step k^*_a.

5. Trent publishes k^*_a as Alice's public key.

6. Trent transmits $m = (S', T', [q_1, \ldots, q_{l'}])$ to Alice, using k_a (see below).

7. Alice modifies her private key to $(S' \circ S, P, T \circ T')$ and stores the polynomials $(q_1, \ldots, q_{l'})$.

8. Trent destroys $S', T', (q_1, \ldots, q_{l'}), k_a$, and k'_a and any results of intermediate computations.

Figure 5.8: Modified Shared Key Generation between Alice and Trent

uses some modifications of HFE, namely the "-" modification (l equations removed, see Section 4.5), the "+" modification (l' added equations, see Section 4.6), the "v" modification (v added variables, see Section 4.7), and the "f" modification (f variables fixed, see Section 4.8). Here Alice generates an HFEv- key (Step 1) while Trent fixes f input variables and also adds l' random equations (steps 3 and 4). Although Step 8 is not necessary for the security of this system, it is recommended. The suggested values in [Co01a, Sec. 21] are $q = 2$, $d = 25$, $n = 167$, $v = 76$, $l = 10$, $l' = 4$, $f = 3$. In this case the overall signature size is $n + v - f = 240$ bits. On a Pentium-III 500 MHz one signature generation takes ≈ 30 seconds [Co01a, Sec. 21].

We are confident that shared key generation as outlined in Figure 5.8

can address the problem of central leakage when the key to be embedded requires a special form. Unfortunately, the algorithm outlined in [Co01a] can not be used without modification. The reason is Step 6 in combination with the recommended values, especially $v = 76$ and $l = 10$. As we saw in Section 4.7, HFEv with a high number of vinegar variables can not be used for encryption. A value of 76 vinegar variables is certainly a high value. Moreover, in Section 4.5 we discussed how to use HFE- for encryption. Again, 10 removed equations are rather high. Combining both numbers, Alice has to test up to $2^{86} \approx 10^{25}$ possible messages to obtain the message sent by Trent in Step 6. As 2^{80} is seen to be a rather high security level (e.g., it is required in the "Call for Cryptographic Primitives" for asymmetric signature schemes in [NESSIE]), we can assume that Alice does not have the means to decipher the message m from Trent. So the algorithm presented in Figure 5.8 has to be modified. We suggest: as Trent does not send sensitive information to Alice, our first solution is not to encrypt the message m at all. If encryption is necessary, Alice and Trent need another algorithm (symmetric or asymmetric). HFEv- with the above parameters is not an option. To have a system which uses only HFE, it might be necessary to have a version of HFE which can be used for encryption.

5.4 eHFE: HFE for Encryption

As we saw in the previous section, some situations may require to encrypt material rather than signing it. Up to the knowledge of the author, HFE has not been used for this purpose yet. In this section, we sketch how HFE might be used for sending a secret (i.e., a symmetric) session key from Alice to Bob. We assume that Bob has a public/private key pair (k_b, K_B) and that Alice has access to a valid copy of Bob's public key k_b.

The overall idea of eHFE (see Figure 5.9) is to combine HFEm' and HFEf. In this figure, $\overline{p_1}, \ldots, \overline{p_n}$ are an intermediate result, depending on the private polynomial P, $q_1, \ldots, q_{m'}$ are randomly generated polynomials, and $p_1, \ldots, p_{m'+n}$ are the overall result for the public key. The HFEf part of eHFE hides the internal structure of the trapdoor to an attacker while the HFEm' step adds some random equations and also some more variables. Both modifications are supposed to enhance the overall security of HFE. For encryption, $h = m + n - f$ elements from $\mathbb{F}$ are used to evaluate $h' = m' + n$ polynomials. For decryption, the trapdoor $K_B = (S, P, T)$ can be used to determinate up to $dq^{m-m'}$ solutions. As outlined in Section 4.9, we can use additional hash bits to pick the correct solution. Possible parameters for

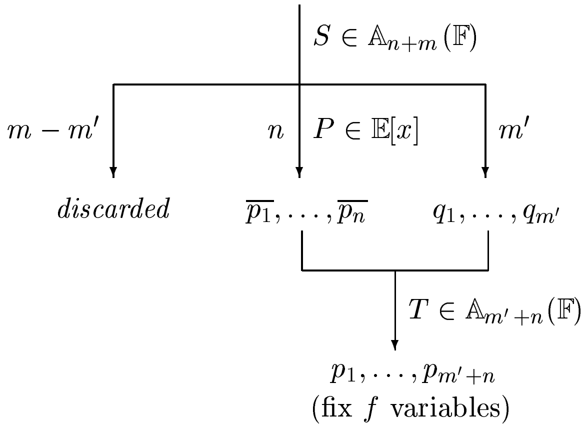

Figure 5.9: HFE for encryption (eHFE)

eHFE-256 (i.e., the session key has a bit size of 256 bits) are field size $q = 2$, extension field dimension $n = 241$ (prime), added variables $m = 19$, added equations $m' = 10$, and fixed variables $f = 4$. For eHFE-128 the corresponding values could be $q = 2$, $n = 113$ (prime), $m = 20$, $m' = 10$, $f = 5$. For example, for eHFE-256 the system needs to test up to $d2^{19} > 500,000d$ solutions. As this requires only one invocation of the root finding algorithm (rather expensive, see Section 2.5), this seems feasible. As the "building blocks" are very new it is unclear at present if such a system would be secure. To answer this question, further work is required. The main question is the degree d of the private polynomial P to obtain a secure system, i.e., more than 2^{128} computations to break eHFE-128 and more than 2^{256} computations to break eHFE-256.

Chapter 6

Conclusions

As we saw in the previous section, HFE is an interesting system which can be adapted to various application domains: from Quartz for long-term security and SFlash for smart cards to HFE for shared key generation. They all look quite different but are a version of the same system: Hidden Field Equations. HFE is based on an NP complete problem, called "multivariable quadratic" (MQ, see Section 3.2).

Although HFE is quite new (it was proposed in 1996 and the latest - successful attack - dates from this year, see Section 3.6), its versions Quartz and SFlash are still under consideration in the NESSIE process. In addition, it is not based on the factorisation problem (see Section 1.2) or the discrete logarithm problem (see Section 1.3) but on the completely different problem of solving a system of quadratic equations over a finite field (MQ). So all algorithms which have been found so far to solve the previous two problems do not apply to MQ. As a result, HFE can provide public key cryptography even if the factorisation problem and the discrete logarithm problem were solved. On the other hand, there has been markable success in finding attacks against MQ. Interestingly, it seems possible to break well-known symmetric ciphers by expressing them as overdefined systems of at most quadratic terms over GF(2) [CoPi02]. Even without these - rather unexpected - applications to symmetric ciphers, HFE itself is still an interesting area of research. For example, it allows public key signature with only 128 bits (see Section 5.2) and its operations are very suitable for hardware and software implementation. This is especially true for signature verification and encryption. However, some questions are still unanswered. Especially the latest attacks against HFE did not provide deeper theoretical understanding about HFE but relay heavily on empirical evidence (see

Section 3.6). To close this gap between empirical results on the one hand and theoretical understanding on the other, more research is necessary. One way to obtain a deeper understanding of HFE may be a rigours count of the number of independent key in HFE, which has been proposed in this thesis (see Section 2.8).

From a more practical perspective, HFE is very interesting as it is rather fast. Especially encryption, not so much decryption (see Section 2.5). Unfortunately, the public key size is rather big (see Section 2.7). Even with these drawbacks, HFE is worthwhile. For example, in a mobile environment with low band width, short signatures are certainly an interesting feature. Imagine that we want to send orders from a mobile phone to a server, which is connected to the Internet. To protect the merchant from fraud, these orders should be signed with a public key cryptographic system. Using HFE for this purpose, we can store the public key outside the mobile phone, e.g., on a server of the mobile phone company, so the drawback of having a big public key is no longer such a big problem as this key (and its certificate) can be sent through the Internet backbone with high band width.

Unfortunately, HFE does not seem to be suitable for encryption. HFEf and HFE+ - which are both suitable for encryption - lead to overdefined systems of equations, i.e., to dangerous situations in terms of recent attacks (see Section 3.6). The new modifications HFEm, HFEm' and HFEz are too new - and due to time limitations for this thesis, there has not been any simulations as, e.g., in [CDF02]. As these simulations sometimes have an unexpected outcome, it is far too risky at the moment to put much effort in using these new modifications for HFE.

As an overall conclusion, HFE is a very interesting research topic. Using HFE for encryption is no option at the moment although it would be worthwhile to have another alternative for public key encryption. In my opinion, the time since the latest attacks (this year) is far too short to use HFE at the moment as the security of HFE is still an open question. On the up side, HFE is a very flexible scheme. Judging from our past experience, it was always possible to change HFE to counter these attacks.

Bibliography

[Bi00] Eli Biham: *Cryptanalysis of Patarin's 2-Round Public Key System with S Boxes (2R)*, Lecture Notes in Computer Science, vol. 1807, pp. 408–, `citeseer.nj.nec.com/article/biham00cryptanalysis.html`

[BSS99] Ian Blake, Gadiel Seroussi, Nigel Smart: *Elliptic Curves in Cryptography*, Cambridge University Press, 1999. ISBN 0-521-65374-6

[CDF02] Nicolas T. Courtois, Magnus Daum, and Patrick Felke: *On the Security of HFE, HFEv- and Quartz*, Cryptology ePrint Archive, Report 2002/13. `http://eprint.iacr.org/2002/138`

[CGP00] Nicolas Courtois, Louis Goubin, and Jacques Patarin: *Submission of Quartz, Flash, and SFlash for NESSIE.* `http://www.cosic.esat.kuleuven.ac.be/nessie/...` `.../workshop/submissions/quartz.zip`, `.../workshop/submissions/flash.zip`, `.../workshop/submissions/sflash.zip`

[CGP01] Nicolas Courtois, Louis Goubin, and Jacques Patarin: *Quartz and SFlash, second revised version.* `www.cosic.esat.kuleuven.ac.be/nessie/...` `.../updatedPhase2Specs/quartz/quartzv21-b.zip` `.../updatedPhase2Specs/sflash/sflashv2.zip`

[Co01] Nicolas T. Courtois: *The security of Hidden Field Equations (HFE)*, Cryptographers' Track Rsa Conference 2001, LNCS2020, Springer-Verlag. `http://www.minrank.org/hfesec.{ps|dvi|pdf}`

[Co01a] Nicolas T. Courtois: *On multivariate signature-only public key cryptosystems.* Cryptology ePrint Archive, Report 2001/029. `http://eprint.iacr.org/2001/029/`

[CoPi02] Nicolas T. Courtois and Josef Pieprzyk: *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations.* Cryptology ePrint Archive: Report 2002/044. `http://eprint.iacr.org/2002/044/`

[Da01] Magnus Daum: *Das Kryptosystem HFE und quadratische Gleichungssysteme über endlichen Körpern*, Diplomarbeit, August 2001, Universität Dortmund. `http://homepage.ruhr-uni-bochum.de/Magnus.Daum/...` `.../HFE.ps.zip` or `.../HFE.pdf`

[DiHe76] Whitfield Diffie, Martin Hellman: *New Directions in Cryptography*, IEEE Transactions on Information Theory, 22 (1976), 644-654.

[Fa02] Jean-Charles Faugère: *HFE Challenge 1 Broken in 96 Hours*, 19th of April 2002, in `news://sci.crypt`

[FrYe79] A.S.Fraenkel and Y.Yesha: *Complexity of problems in games, graphs and algebraic equations*, Discrete Applied Mathematics, 1 (1979), 15-30

[GaJo79] 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

[GSB01] W. Geiselmann, R. Steinwandt, and Th. Beth: *Attacking the Affine Parts of SFLASH*, Cryptography and Coding, 8th IMA International Conference Proceedings, B. Honary, ed., vol. 2260 of Lecture Notes in Computer Science, pp. 355-359, Springer, 2001. *Also presented at the 2nd NESSIE workshop.*

[HFE] Nicolas T. Courtois: *Hidden Field Equations public key cryptosystem home page (HFE)*, `http://www.minrank.org/hfe/`

[Hi96] Richard O. Hill: *Elementary Linear Algebra with Applications*, Harcourt College Publishers, 1996. ISBN 0-03-010347-9

[KiSh99] Aviad Kipnis and Adi Shamir: *Cryptanalysis of the HFE public key cryptosystem*, Crypto'99, `http://www.minrank.org/hfesubreg.ps` or `http://citeseer.nj.nec.com/kipnis99cryptanalysis.html`

[LiNi86] Rudolf Lidl and Harald Niederreiter: *Introduction to finite fields and their applications*, Cambridge University Press, 1986. ISBN 0-521-30706-6

[Ma01] Gwenaëlle Martinet: *Quartz, Flash and SFlash*, Report for NESSIE, Document NES\DOC\ENS\WP3\006\2, 7th of March 2001, `http://www.cosic.esat.kuleuven.ac.be/...` `.../nessie/reports/enswp3-006-2.pdf`

[Ma01a] Gwenaëlle Martinet: *Selecting a* $C**--$*scheme*, Report for NESSIE, Document NES\DOC\ENS\WP3\009\a, `http://www.cosic.esat.kuleuven.ac.be/...` `.../nessie/reports/enswp3-009a.pdf`

[MaIm88] Tsutomu Matsumoto and Hideki Imai: *Public Quadratic Polynomial-Tuples for Efficient Signature Verification and Message-Encryption*, Eurocrypt'88.

[Me+93] Alfred J. Menezes et al.: *Applications of finite fields*, Kluwer Academic Publishers Group, 1993. ISBN 0-7923-9282-5

[Me+96] Alfred J. Menezes et al.: *Handbook of Applied Cryptography*, CRC Press, 1996, ISBN 0-8493-8523-7

[NESSIE] Homepage of the NESSIE project: `http://www.cosic.esat.kuleuven.ac.be/nessie/`

[NIST01] National Institute of Standards and Technology: *Federal Information Processing Standards Publication 180-2*, August 1, 2001. `http://csrc.nist.gov/publications/fips/...` `.../fips180-2/fips180-2.pdf`

[Pa95] Jacques Patarin: *Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88*, Crypto '95, pp. 248–261,

[Pa96] Jacques Patarin: *Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): two new Families of Asymmetric Algorithms*, EuroCrypt '96. *Extended Version*: http://www.minrank.org/hfe.pdf

[PaGo97] Jacques Patarin and Louis Goubin: *Trapdoor One-Way Permutations and Multivariate Polynomials - Extended Version*, Proc. 1st International Information and Communications Security Conference (1997), pp. 356–368, `citeseer.nj.nec.com/patarin97trapdoor.html`

[PaGo97a] Jacques Patarin and Louis Goubin: *Asymmetric Cryptography with S-Boxes*, Proc. 1st International Information and Communications Security Conference (1997), pp. 369–380, `http://citeseer.nj.nec.com/patarin97asymmetric.html`

[PGC98] Jacques Patarin, Louis Goubin, Nicolas Courtois: *C^*_{-+} and HM: Variations around two schemes of T.Matsumoto and H.Imai*, 1998, `citeseer.nj.nec.com/patarin98plusmn.html`

[Pr+02] B. Preneel, A. Bosselaers, S.B. Örs, et. al: *Update on the selection of algorithms for further investigation during the second round*, Document NES/DOC/ENS/WP5/D18/1, March 2002, `http://www.cosic.esat.kuleuven.ac.be/...` `.../nessie/deliverables/D18_1.pdf`

[SCPK00] Adi Shamir, Nicolas Courtois, Jacques Patarin, and Alexander Klimov: *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*, Eurocrypt 2000. `http://www.minrank.org/xlfull.pdf`

[Sc96] Bruce Schneier: *Applied Cryptography - 2^{nd} Edition: protocols, algorithms, and source code in C*, John Wiley & Sons, Inc., 1996. ISBN 0-471-12845-7 or 0-471-11709-9.

[Sh97] P. Shor: *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal on Computing, 26(5):1484–1509, 1997.

[Wo01] Christopher Wolf: *Hidden Field Equations (HFE)*, Project CS4095 at University College Cork, `http://www.christopher-wolf.de/hfe` `-> project report`

Erklärung

Hiermit erkläre ich, dass ich diese Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Neu-Ulm, den 2. Dezember 2002