A Fast and Secure Implementation of Sflash[⋆]

Mehdi-Laurent Akkar, Nicolas T. Courtois, Romain Duteuil, and Louis Goubin

SchlumbergerSema, CP8 Crypto Lab
36-38 rue de la Princesse, 78430 Louveciennes, France
{MAkkar,NCourtois,RDuteuil,LGoubin}@slb.com

Abstract. Sflash is a multivariate signature scheme, and a candidate for standardisation, currently evaluated by the European call for primitives Nessie. The present paper is about the design of a highly optimized implementation of Sflash on a low-cost 8-bit smart card (without coprocessor). On top of this, we will also present a method to protect the implementation protection against power attacks such as Differential Power Analysis.
Our fastest implementation of Sflash takes 59 ms on a 8051 based CPU at 10MHz. Though the security of Sflash is not as well understood as for example for RSA, Sflash is apparently the fastest signature scheme known. It is suitable to implement PKI on low-cost smart card, token or palm devices. It allows also to propose secure low-cost payment/banking solutions.

Keywords: Digital Signatures, PKI, Addition Chains, Multivariate Cryptography, Matsumoto-Imai cryptosystem C^*, C^{*--} trapdoor function, HFE, portable devices, Smart cards, Power Analysis, SPA, DPA.

1 Introduction

The design of Flash and Sflash signature schemes is due to Courtois, Patarin and Goubin [17, 18]. Sflash is based on a so called C^{*--} multivariate signature scheme, the name of C^{*--} being due to Patarin [14]. The idea goes back to the Matsumoto-Imai cryptosystem proposed at Eurocrypt'88, also called C^* in [12], that is a remote cousin of RSA, that uses a power function over an extension of a finite field. At the time the Matsumoto-Imai or C^* cryptosystem was believed very secure, and were amazingly fast. At the same time, in France, the smart cards become a great success: they allow to divide by ten the fraud figures in the payment systems. However RSA is too slow to be used on a smart card, and this keeps the security achieved by smart cards solutions insufficient: unable to implement a real public key signature. From this arises the motivation to find a signature scheme that could be implemented on low-cost smart cards. Unfortunately, at Crypto'95 Patarin shows Matsumoto-Imai is insecure[1], see [13, 10]. Subsequently Patarin studied many other variants and

[⋆] The work described in this paper has been supported by the French Ministry of Research under RNRT Project "Turbo-signatures".
[1] Note that H. Dobbertin claims to have independently found this attack in 93/94.

generalizations of Matsumoto-Imai (or C^*) (for example the Dragons). Most of them are broken, and very few remain. Among these, C^{*-} is particularly simple, and remains unbroken. It is simply the original scheme combined with the idea of preventing structural attacks by simply removing some of the equations that constitute the public key, due initially to Shamir [19]. At Asiacrypt'98 [14], Courtois, Patarin and Goubin show that C^{*-} can be attacked, and if r is the number of removed equations, a factor of q^r appears in the attack. For various reasons it is conjectured that the security of C^{*-} does increase with at least q^r when removing equations [1], and the same is also conjectured for other multivariate cryptosystems[2]. When q^r is very big, e.g. 2^{80}, it is believed that C^{*-} is secure, we then call it C^{*--}, as a lot of equations are removed. It is possible to see that due to many equations removed, C^{*--} can only be used in signature, no longer in encryption.

From C^{*-}, in 2000, Courtois, Patarin and Goubin designed the Flash signature scheme, submitted to Nessie European call for cryptographic primitives, and also a special version of Flash called Sflash that manages to decrease the size of the public key[3]. Unfortunately at Eurocrypt'2002, Gilbert and Minier, showed that this very trick, used to decrease the size of the public key of Sflash, is insecure, and broke Sflash, see [6]. Since then the specification of Sflash has been revised, the new version of Sflash is in fact a version of Flash with a better choice of parameters.

Finally, it is important to note that the design of Flash and Sflash does not reduce to C^{*--}. There is difficulty in the design of multivariate signature schemes that comes from the fact that the systems of equations, have in general many solutions, and only the knowledge of the internal algebraic structure allows to find one of them, which is usually done by fixing some internal variables. If this process is not handled correctly, it might leak information about this internal structure, and eventually allow to recover the private key of Flash/Sflash. See [17] and [1].

Sflash is therefore the best solution that has been found so far to make digital signatures in a low-cost device such as smart card, USB token or a palm device. However, the security of an implementation of a cryptographic algorithm in such a device does not reduce to the security of the cryptographic algorithm itself. It is hard to protect a secret that is entirely in the hands of a potential attacker: the implementation should also have in mind possible side-channel attacks. In 1998, Kocher, Jaffe and Jun showed the feasibility of such attacks [11] using the power consumption of the device, and since then other side-channel attacks have been proposed. In this paper we will also describe, on top of our optimized implementation, a protection against side-channel attacks.

[2] For example see the experiments with Buchberger's algorithm applied to HFE-, presented in [2] (in these proceedings) or on slide 35 of [3].

[3] The main drawback of many multivariate cryptosystems.

2 Structure of the Smart Card

We consider a low-end smart card built on a 8-bit CPU core, an Intel 8051. It has no arithmetic or cryptographic coprocessor. The memory of this card is divided in three parts as follows:

- The *data*: 128 bytes which one can address directly in one CPU clock.
- The *xdata*: between 1 and 4 Kbytes, indirectly addressable in two CPU clocks.
- The *code*: between 4 and 64 Kbytes of unrewritable memory, indirectly addressable in two CPU clocks.
- Most smart card processors that are based on 8051 contain also between 2 and 128 Kkbytes of E^2PROM memory, that can be used to store keys.

As a comparison one could see the *data*, the *idata* and the *xdata* like the RAM of a classical PC, whereas the *code* would be the ROM. According to those considerations, we will try to store the most manipulated variables in the *data* in order to save as much time as possible in the computation.

3 Basis Structures and Variables Used in Sflash

A complete description of Sflash can be found in [17, 18]. The signature process consists mainly of a composition of three functions defined over the extensions of finite fields: $s^{-1} \circ f^{-1} \circ t^{-1}$, with two multivariate affine functions s, t and one univariate power function f, In this paper we concentrate on the implementation of the basic operations that are used in Sflash.

3.1 Main Structures

The algorithm mainly manipulates elements of the finite field $L = GF(128^{37})$, constructed as an extension of the base Galois field $K = GF(128)$. The field $K = GF(128)$ defined as $GF(2)[X]$ polynomials modulo $(X^7 + X + 1)$:

$$K = GF(2)[X]/(X^7 + X + 1)$$

Each element of K is written as 7 bits stored in one byte; the coefficient of X^i becomes the coefficient of 2^i, $i = 0..6$. The big field $L = GF(128^{37})$ is defined as follows:

$$L = K[X]/(X^{37} + X^{12} + X^{10} + X^2 + 1) \quad (1)$$

We also identify L with K^{37} and represent an element of L by 37 elements of K (the coefficients of X^i, $i = 0..36$), that in turn are written as 37 bytes, each of them using only 7 bits.

Our implementation uses two temporary variables called y and *temp*, that are structures containing an element of L, in the *data* zone of the smart card. This allows them to be easily accessible so that the computation is faster. We will also store another structure of type L, called x, in the *xdata* zone because at some moment we need to manipulate three elements of L, when using a function having two inputs and one output (we are not able to store this third one in the *data* as we need additional space in the *data* zone for something else).

3.2 The Private Key of Sflash

The secret parameters of the signature scheme are the two transformations from L to L, s and t, that are multivariate affine functions, *i.e.* they are affine when seen as functions from $K^{37} \to K^{37}$. We do not actually store s and t but their inverses s^{-1} and t^{-1}. We have $t^{-1} : x \mapsto T_m x + T_c$ and $s^{-1} : x \mapsto S_m x + S_c$ with T_m and S_m being two matrixes 37×37 and with T_c and S_c being the constant vectors[4]. All these are stored in either the *code* zone or the E^{2}PROM of the card.
We will also store the 80-bit secret string Δ in the *code* zone (or E^{2}PROM).

4 Fast Implementation of the Operations over the Fields

The Implementation of K:

1. Addition: Easy in characteristic 2 of $K = GF(128)$, the addition in $K = GF(128)$ is implemented by XORing the byte representations of the elements.
2. Multiplication: more work is required here. Since the multiplicative group K^* is cyclic, say generated by α, each element of K (but zero) can be seen as a power of α. Powers add when we multiply two non-zero elements: $\alpha^x.\alpha^y = \alpha^{x+y}$, and the multiplication by zero is obvious. In order to execute this operation, we store two tables of 127 bytes, one (named *expo* and stored in the *code* zone) giving the exponent of α corresponding to a given non-zero element of K, and the reciprocal operation (named *log* and also in *code*) giving the element of K corresponding to a given exponent of α.

The Implementation of L: Based on the definition of L in (1).

1. Addition: given two elements of L represented by two polynomials $a = \sum_{i=0}^{36} a_i X^i$ and $b = \sum_{i=0}^{36} b_i X^i$, their sum is computed by XORing the coefficients of the same degree:
$$c \ = \ a + b \ \overset{def}{=} \ \sum_{i=0}^{36} (a_i \oplus b_i) X^i$$
with $\oplus$ being the XOR operation.
2. Multiplication: It is costly, because we compute the product of two polynomials a and b which will be of degree 72, and then compute its euclidian reduction modulo the irreducible polynomial $X^{37} + X^{12} + X^{10} + X^2 + 1$ (as there are no trinomials for this field).
   (a) Build $c' = \sum_{i=0}^{72} c'_i X^i$, where $c_i = \sum_{l+k=i} a_l.b_k$.
   (b) Then reduce c' by $X^{37} + X^{12} + X^{10} + X^2 + 1$, the result c is the product $a.b$ in L.

[4] Actually these two constant vectors are not really secret as shown in [8]. Therefore one can choose s and t linear instead of affine, which reduces the size of the secret key.

3. Square: It is interesting to code the squaring operation in L independently, it can be done at least 5 times faster than a multiplication of an element by itself. The above multiplication algorithm requires a quadratic-time computation on the coordinates of the two operands (the building of c' in (a) above) whereas, to square an element a, we only have to compute:

$$a' = \sum_{i=0}^{36} a_i^2 X^{2i}$$

which is linear in the a_i, and then reduce a' modulo $X^{37}+X^{12}+X^{10}+X^2+1$, like in step (b) above.

The Affine Transformations s and t: They are computed as classical matrix multiplications, with additional XOR with the constant vector. As in each step of the matrix multiplication we have to compute several multiplications in K (one of the coordinates of the input with one of the matrix' coefficients), and regarding to how we compute such a multiplication (*cf* above), it will be faster if we store base α logarithms of the coefficients of the matrix.

5 How to Compute $A = f^{-1}(B)$ in Sflash ?

We need to compute $A = B^h$ in L, with:

$$\begin{aligned} h &= (128^{11}+1)^{-1} \mod (128^{37}-1) \\ &= 1000000\ 1000000\ 1000000\ 0111111\ 0111111\ 0111111\ 0111111\ 1000000 \\ &\quad\ 1000000\ 1000000\ 1000000\ 0111111\ 0111111\ 0111111\ 1000000\ 1000000 \\ &\quad\ 1000000\ 1000000\ 0111111\ 0111111\ 0111111\ 0111111\ 1000000\ 1000000 \\ &\quad\ 1000000\ 0111111\ 0111111\ 0111111\ 0111111\ 1000000\ 1000000\ 1000000 \\ &\quad\ 1000000\ 0111111\ 0111111\ 0111111\ 1000000 \end{aligned}$$

The cost of a classical "square and multiply" algorithm to carry on this power h would be, at least, 259 squaring and 145 multiplications ! The slowest operation is the multiplication in the field L (squarings are faster). Our best implementation of multiplication in L requires about 10 000 CPU cycles (which takes at least 2.4 ms on a 10 MHz smart card).

We need to find an addition chain for h involving as little multiplications as possible. We did not limit ourselves to finding a "classical" addition chain for h, but also privileged some special powers, namely 128 and 128^7 which, as we will see in the next part, are quite easy to compute. After numerous attempts we have found the following method:

- $\alpha \longleftarrow (B^2)^2$;
- $\beta \longleftarrow B \times \alpha$;
- $\gamma \longleftarrow (\alpha^2)^2$;
- $\delta \longleftarrow \beta \times \gamma$;
- $u \longleftarrow \delta^2 \times \delta$;
- $v \longleftarrow (\gamma^2)^2$;

- $t \longleftarrow ((v^{128})^{128})^{128} \times u$;
- $w \longleftarrow ((t^{128} \times t)^{128} \times t)^{128}$;
- $x \longleftarrow ((w \times u)^{128})^{128^7}$;
- $z \longleftarrow v^{128^7} \times w \times v \times x$;
- $A \longleftarrow (((z^{128^7})^{128^7})^{128} \times x)^{128^7} \times z$.

This method has a particularly low number of multiplications: 12, instead of 145 for "square and multiply".

5.1 Special Operations Involved in the Computation of f^{-1}:

We already described the implementation of the multiplication and the squaring. What remain, are the efficient implementations of $x \mapsto x^{128}$ and $x \mapsto x^{128^7}$. In $K = GF(128)$, those two operations are K-linear on L, so they can be seen fulfilled with a matrix multiplication. Moreover, due to the fact that the polynomial by which we reduced $(X^{37} + X^{12} + X^{10} + X^2 + 1)$ has only coefficients 0 and 1, the matrices involved are only made up of 0s and 1s, which allows us to store their coefficients on single bits.

The Function $\mathbf{x} \mapsto \mathbf{x^{128^7}}$: As the matrix used for the raising to the power 128^7 is fixed, we can also accelerate this computation by finding a "cheap" road in the matrix, which is related to the idea of the Gray code. The Gray code is an ordering of all binary n-bits words, such two consecutive words differ by only one bit, see [9] for example. This allows to compute all possible linear combinations over $GF(2)$ of some n vectors, with one XOR by combination, instead of $n/2$. We use an even better, specific solution that is adapted to the particular matrix that is fixed (even for two different private keys). Our goal is to compute:

$$y = M.x \;;\quad \text{with } x, y \in K^{37} \text{ and } M \in \mathbf{M}_{37,37}(GF(2))$$

For this we divide the matrix in some $37 \times n$ submatrices for some small n. For each submatrix we look for a "cheap" road: each y_i is a XOR of different x_i, how to compute them minimizing the number of XORs as possible. For example we assume that the first submatrix begins with:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ .. \end{pmatrix} = \begin{pmatrix} 0\,0\,1\,0\,1\,0\,0 \\ 0\,0\,1\,1\,1\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,1\,1\,0\,0\,0 \\ .\;.\;.\;.\;.\;.\;.\; \end{pmatrix} \cdot \begin{pmatrix} x_1\; x_2\; x_3\; x_4\; x_5\; x_6\; x_7 \ldots \end{pmatrix}^T$$

We store $x_1, ..., x_n$ in separate registers. Let A be the main register. We first put $A \leftarrow x_5$. Then we XOR A with x_3: $A \leftarrow A \oplus x_3$ and then put $y_1 \leftarrow A$, now $y_3 = x_3 \oplus x_5$. Then we put $y_2 \leftarrow A \leftarrow A \oplus x_4$. Finally we do $y_4 \leftarrow A \leftarrow A \oplus x_5$, etc.. A "cheap" road should use about 1 XOR per y_i computed. We need to find the cheapest road in each submatrix and also to find the best n for such an operation. For our specific matrices 37×37 $n = 7$ seemed to be optimal and our best solution has been found with some computer simulations.
This technique allows to accelerate quite a lot (about 40 times !) the operation $\mathbf{x} \mapsto \mathbf{x^{128^7}}$, however it takes a lot of space in term of program code (0.7 Kbyte).

The Function $\mathbf{x} \mapsto \mathbf{x^{128}}$: It is useless to use the above technique to compute the power $128 = 2^7$. Doing 7 successive squarings is faster than a matrix multiplication, even if done in a clever way. It is also much cheaper in term of code size, not only we have no matrix to store, but we will mostly re-use the existing code. In addition to that, $\mathbf{x} \mapsto \mathbf{x^{128}}$ can be computed faster than doing seven squarings in a row. Indeed, seven squares will be a succession of squares in $GF(128)$ done position by position on 37 values, and a multivariate linear operation over $GF(128)^{37}$ that comprises expansion and modular reduction modulo the irreducible polynomial. It is easy to see that "position by position" squaring commutes with the multivariate linear part. Thus we may postpone all the 7 "position by position" squarings to the end, and then we realize that they are not needed, because in $GF(128)$ we have always $x^{128} = x$.

6 The Performance Data

To summarize, our implementation of Sflash requires:
- 2 matrix products to apply T_m and S_m.
- 12 multiplications in L.
- 7 squarings in L.
- 8 raisings to the power 128 in L.
- 5 raisings to the power 128^7.

From here we have two possible implementations of Sflash:

- A **fast** one, using the technique above to compute the power 128^7 which is quite fast but a bit large in term of code size:
  - RAM: 334 bytes (112 bytes of *data* and 222 bytes of *xdata*).
  - Code size (ROM): 3.1 Kbytes.
- A **slower** one, using a classical matrix product which is slower (but still acceptable) and have a shorter code:
  - RAM: 334 bytes (112 bytes of *data* and 222 bytes of *xdata*).
  - Code size (ROM): 2.5 Kbytes.

We have implemented the two versions of Sflash on two Intel 8051 CPU based components: an original Intel 8051 CPU and an Infineon SLE66 component without cryptoprocessor. The timings (without hashing) are the following:

Component	8051	8051	SLE66	SLE66	8051	8051	SLE66	SLE66
Frequency [MHz]	3.57	10	3.57	10	3.57	10	3.57	10
Code version	fast	fast	fast	fast	slow	slow	slow	slow
ROM size [kbytes]	3.1	3.1	3.1	3.1	2.5	2.5	2.5	2.5
Timings [ms]	750	268	164	**59**	1075	384	230	82

We see that on a smart card without any coprocessor and in usual conditions (10 MHz is today a normal frequency for a low-cost component such as SLE66) one can compute digital signatures in as little as 59 ms ! It is much less than for RSA or Elliptic Curves, even if a cryptographic co-processor is used (!), see the comparison in Section 8.

7 Protecting Sflash against Side-Channel Attacks

In the side-channel attacks, the adversary tries to recover the private key of a signature scheme (or other useful information) given the information that leaks from the intermediate data that appears during the computation, or from the computation itself, see for example [11].

7.1 Protecting against SPA-Like Attacks

For Sflash, the computation is always the same, whatever the value of the private key is. Moreover, all the computations in Sflash use full bytes of the private key, making the dependency of power consumption on the key very complex to exploit. This prevents SPA attacks known for unprotected implementations of RSA, in which the power consumption allows to see the values of single bits of the private key.

7.2 Protecting against DPA-Like Attacks

For DPA-like attacks, the protection boils down to masking the intermediate data. The signature of Sflash involves mainly the composition of 3 functions, t^{-1}, f^{-1} and s^{-1}. The best way to prevent an information leak (of any type) is to mask completely the two intermediate values: the output of t^{-1} and of f^{-1}. For this, we will use the homomorphic properties of the functions t^{-1} and f^{-1} with regard to respectively the addition and the multiplication. Thus, since t^{-1} is affine, its output is still masked additively (with another mask). Similarly we may pass through f^{-1} with a multiplicative masking.

The proposed method for a secured implementation of Sflash is shown on Figure (1).

7.3 Algorithmic Considerations

A. Computation of $f(\lambda)$: The function f is the following :

$$f : x \mapsto x^{128^{11}+1} = \left(((((x^{128^7})^{128})^{128})^{128})^{128} \right) \cdot x$$

so we can compute $f(\lambda)$ with one raising to the power 128^7, 4 raisings to the power 128 and one multiplication.

B. Computation of λ^{-1}: First of all we begin with remarking that $|L^*| = 128^{37} - 1$, so inverting an element of L can be done by raising it to the power $128^{37} - 2$. Besides $128^{37} - 2 = (11...110)_2$ (*i.e.* 258 "1" followed by "0" in basis 2). Thus we can compute λ^{-1} as follows:

- $z \longleftarrow \lambda$
- $z \longleftarrow z^2.z$ (i.e. $z = \lambda^{(11)_2}$ that we store)
- $z \longleftarrow z^{2^2}.z$ ($z = \lambda^{(1111)_2}$)

$x \longleftarrow$ hashed message to sign.
$r \longleftarrow$ random $\in L$.
$\lambda \longleftarrow$ random $\in L^*$.

$$\begin{tikzcd}
x \arrow[r] & (x \oplus r) \arrow[d, "t^{-1}"'] & & r \arrow[d, "\times\ T_m"] & & \\
& t^{-1}(x) \oplus T_m r \arrow[dr, "\times\ f(\lambda)"'] & & T_m r \arrow[dl, "\times\ f(\lambda)"] & & \\
& & \oplus \arrow[d] & & & \\
& & f(\lambda)\ t^{-1}(x) \arrow[d, "f^{-1}"'] & & & \\
& & \lambda\ (f^{-1} \circ t^{-1}(x)) & \oplus \arrow[d, "\times\ \lambda^{-1}"] & \lambda\ r & \\
& & & f^{-1} \circ t^{-1}(x) \oplus r \arrow[dr, "s^{-1}"'] & & r \arrow[dl, "\times\ S_m"] \\
& & & & \oplus \arrow[d] & \\
& & & \text{Signature of } x: & s^{-1} \circ f^{-1} \circ t^{-1}(x) &
\end{tikzcd}$$

Fig. 1. A randomized masking method to protect Sflash against side-channel attacks

- $z \longleftarrow z^{2^4}.z$ $(z = \lambda^{(11111111)_2})$
- $z \longleftarrow (z^{128})^2.z$ $(z = \lambda^{(1111111111111111)_2})$
- $z \longleftarrow ((z^{128})^{128})^4.z$ $(z = \lambda^{(11111111111111111111111111111111)_2})$
- $z \longleftarrow ((((z^{128})^{128})^{128})^{128})^{2^4}.z$ $(z = \lambda^{(11...11)_2}$ with 64 "1")
- $z \longleftarrow z^{2^{64}}.z = (((z^{128^7})^{128})^{128})^2.z$ $(z = \lambda^{(11...11)_2}$ with 128 "1")
- $z \longleftarrow z^{2^{128}}.z = ((((z^{128^7})^{128^7})^{128})^{128})^{128})^{128})^4.z$
  (now we have $z = \lambda^{(11...11)_2}$ with 256 "1")
- $z \longleftarrow z^{2^2}.\lambda^{(11)_2}$ $(z = \lambda^{(11...11)_2}$ with 258 "1")
- $z \longleftarrow z^2$ which gives indeed $z = \lambda^{-1} = \lambda^{(11...110)_2}$ with 258 "1" and one "0".

C. Computation of $T_m r$ et $S_m r$: We compute them with a classical matrix product.

Computations Added Compared to an Unprotected Version:
- 2 matrix products to compute $T_m r$ and $S_m r$.
- 13 multiplications in L.
- 20 squarings in L.
- 17 raisings to the power 128 in L.
- 4 raisings to the power 128^7 in L.

Comparing to what we achieved with our (unprotected) implementation of Sflash, see the results in Section 6, the version which is protected against DPA-like attacks implies to approximately double the running time.

8 Digital Signatures on a Smart Card – a Comparison

In this section we compare Sflash to some other known implementations of the signature schemes in smart cards. The numerical data for schemes other than Sflash are based on unverified claims of the vendors.

cryptosystem	Sflash	NTRU-251	RSA-1024	RSA-1024	ECC-191
platform	SLE-66	Philips 8051	SLE-66	ST-19XL	SLE-66
word size [bits]	8	8	8	8	8
ROM size [Kbytes]	3.1	5			
speed [MHz]	10	16	10	10	10
co-processor	no	no	no	yes	yes
Signature Length	259	1757	1024	1024	382
Timings	59 ms	160 ms	many s	111 ms	180 ms

Apparently, the only signature scheme known that might be able to compete with Sflash is NTRUSign. An NTRUSign signature seems to be slower, but also about 6 times longer. Knowing that the communication ports of low-end smart cards are quite slow, at 9600 bit/s, an NTRUSign card would take additional 200 ms to transmit the signature.

9 Conclusion

In this paper we described a highly optimized implementation of the Sflash signature schemes on a low-cost smart card. Our fastest implementation of Sflash takes 59 ms on a 8051 based CPU at 10MHz. We also presented a method to protect this implementation against DPA-like attacks that requires about twice as much time.

Though the security of Sflash is not as well understood as for example for RSA, Sflash is apparently the fastest signature scheme known. It is suitable to implement PKI on low-cost smart card, token or palm devices. It allows also to propose secure low-cost payment/banking solutions.

References

[1] Nicolas Courtois, *La sécurité des primitives cryptographiques basées sur les problèmes algébriques multivariables MQ, IP, MinRank, et HFE*, PhD Thesis, Paris 6 University, 2001, in French. Available at `http://www.minrank.org/phd.pdf` 268

[2] Nicolas Courtois, Magnus Daum, Patrick Felke, *On the Security of HFE, HFEv- and Quartz,* PKC'2003, to appear in LNCS, Springer. 268

[3] Magnus Daum, Patrick Felke, *Some new aspects concerning the Analysis of HFE type Cryptosystems,* Presented at Yet Another Conference on Cryptography (YACC'02), June 3-7, 2002, Porquerolles Island, France. 268

[4] Magnus Daum, *Das Kryptosystem HFE und quadratische Gleichungssysteme über endlichen Körpern,* Diplomarbeit, Universität Dortmund, 2001. Available at `daum@itsc.ruhr-uni-bochum.de`

[5] Jean-Charles Faugère, *Report on a successful attack of HFE Challenge 1 with Gröbner bases algorithm F5/2*, announcement that appeared in `sci.crypt` newsgroup on the internet on April 19th 2002.

[6] Henri Gilbert, Marine Minier, *Cryptanalysis of Sflash,* EUROCRYPT'2002, LNCS 2332, Springer, pp. 288-298. 268

[7] Michael Garey, David Johnson, *Computers and Intractability, a guide to the theory of NP-completeness*, Freeman, p. 251.

[8] Willi Geiselmann, Rainer Steinwandt, Thomas Beth, *Revealing 441 Key Bits of SFLASH-v2*, Third NESSIE Workshop, November 6-7, 2002, Munich, Germany. 270

[9] A page about the Gray code, `http://www.nist.gov/dads/HTML/graycode.html` 272

[10] Neal Koblitz, *Algebraic aspects of cryptography*, Springer, ACM3, 1998, Chapter 4: "Hidden Monomial Cryptosystems", pp. 80-102. 267

[11] Paul Kocher, Joshua Jaffe, Benjamin Jun, *Introduction to Differential Power Analysis and Related Attacks*. Technical Report, Cryptography Research Inc., 1998. Available at `http://www.cryptography.com/dpa/technic/index.html` 268, 274

[12] Tsutomu Matsumoto, Hideki Imai, *Public Quadratic Polynomial-tuples for efficient signature-verification and message-encryption*, EUROCRYPT'88, LNCS 330, Springer 1998, pp. 419-453. 267

[13] Jacques Patarin, *Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88*, CRYPTO'95, LNCS 963, Springer, pp. 248-261. 267

[14] Jacques Patarin, Nicolas Courtois , Louis Goubin, *C*-+ and HM - Variations around two schemes of T. Matsumoto and H. Imai*, ASIACRYPT'98, LNCS 1514, Springer, pp. 35-49. 267, 268

[15] Jacques Patarin, Louis Goubin, Nicolas Courtois, *Quartz,* **1**28-*bit long digital signatures*, Cryptographers' Track RSA Conference 2001, San Francisco 8-12 April 2001, LNCS 2020, Springer, pp. 282-297.
Note: The Quartz signature scheme has been updated since, see [16].

[16] Jacques Patarin, Louis Goubin, Nicolas Courtois, *Quartz,* **1**28-*bit long digital signatures*, An updated version of Quartz specification. available at `http://www.cryptosystem.net/quartz/` or `http://www.cryptonessie.org` 278

[17] Jacques Patarin, Louis Goubin, Nicolas Courtois, *Flash, a fast multivariate signature algorithm*, Cryptographers' Track RSA Conference 2001, San Francisco 8-12 April 2001, LNCS 2020, Springer, pp. 298-307. 267, 268, 269

[18] An updated version of Sflash specification. Available at `http://www.cryptosystem.net/sflash/` or `http://www.cryptonessie.org` 267, 269

[19] Adi Shamir, *Efficient signature schemes based on birational permutations*, CRYPTO'93, LNCS 773, Springer, pp. 1-12. 268