SFLASH, a fast asymmetric signature scheme for low-cost smartcards

Primitive specification and supporting documentation

Nicolas Courtois, Louis Goubin, Jacques Patarin

CP8 Crypto Lab, SchlumbergerSema, 36-38 rue de la Princesse,
BP 45, 78430 Louveciennes Cedex, France
courtois@minrank.org, LGoubin@slb.com, JPatarin@slb.com

Note: This document specifies the updated final version of the SFLASH signature scheme, slightly modified as allowed in the second stage of Nessie evaluation process, in order to improve the speed and the security. This is therefore the only official version of SFLASH. In some papers that refer to the old version, it is sometimes called SFLASH^{v1}, and SFLASH^{v2} is the new version. In the appendix of the present document we summarize all the changes in SFLASH, for readers and developers that are acquainted with the previous version.

1 Introduction

In the present document, we describe the SFLASH public key signature scheme.

SFLASH is a C^{*--} algorithm (see [2]) with a special choice of the parameters. SFLASH belongs to the family of "multivariate" public key schemes, *i.e.* each signature and each hash of the messages to sign are represented by some elements of a small finite field K.

SFLASH is designed to be a very fast signature scheme, both for signature generation and signature verification. It is much faster in signature than RSA and much esier to implement on smartcards without any arithmetic coprocessor for example. However its public key size is larger than the public key size of RSA. Nevertheless this public key size can fit in current smartcards. It may also be noticed that, with the secret key, it is possible to sign AND to check the signature (generated with this particular secret key) without the need of the public key (in some applications this may be useful).

As a result, the parameters of SFLASH have been chosen in order to satisfy an extreme property that no other standardized public key scheme has reached so far: efficiency on low-price smartcards. SFLASH has been specially designed for very specific applications because we thought that for all the classical applications of signature schemes, the classical algorithms (RSA, Fiat-Shamir, Elliptic Curves, DSA, etc) are very nice, but when we need some very specific properties these algorithms just can not satisfy them, and it creates a real practical need for algorithms such as SFLASH.

SFLASH was designed to have a security level of 2^{80} with the present state of the art in Cryptanalysis, as required in the NESSIE project.

2 Notations

In all the present document, || will denote the "concatenation" operation. More precisely, if $\lambda = (\lambda_0, \ldots, \lambda_m)$ and $\mu = (\mu_0, \ldots, \mu_n)$ are two strings of elements (in a given field), then $\lambda||\mu$ denotes the string of elements (in the given field) defined by:

$$\lambda||\mu = (\lambda_0, \ldots, \lambda_m, \mu_0, \ldots, \mu_n).$$

For a given string $\lambda = (\lambda_0, \ldots, \lambda_m)$ of bits and two integers r, s, such that $0 \leq r \leq s \leq m$, we denote by $[\lambda]_{r \to s}$ the string of bits defined by:

$$[\lambda]_{r \to s} = (\lambda_r, \lambda_{r+1}, \ldots, \lambda_{s-1}, \lambda_s).$$

3 Parameters of the algorithm

The SFLASH algorithm uses three finite fields.

- The first one, $K = \mathbf{F}_{128}$ is precisely defined as $K = \mathbf{F}_2[X]/(X^7 + X + 1)$. We will denote by π the bijection between $\{0,1\}^7$ and K defined by:

$$\forall b = (b_0, \ldots, b_6) \in \{0,1\}^7,\ \pi(b) = b_6 X^6 + \ldots + b_1 X + b_0 \pmod{X^7 + X + 1}.$$

- The second one is $\mathcal{L} = K[X]/(X^{37} + X^{12} + X^{10} + X^2 + 1)$. We will denote by φ the bijection between K^{37} and $\mathcal{L}$ defined by:

$$\forall \omega = (\omega_0, \ldots, \omega_{36}) \in K^{37},\ \varphi(\omega) = \omega_{36} X^{36} + \ldots + \omega_1 X + \omega_0 \pmod{X^{37} + X^{12} + X^{10} + X^2 + 1}.$$

3.1 Secret Parameters

1. An affine secret bijection s from K^{37} to K^{37}. Equivalently, this parameter can be described by the 37×37 square matrix and the 37×1 column matrix over K of the transformation s with respect to the canonical basis of K^{37}. We denote by S_L the square matrix ("L" means "linear") and S_C the column matrix (here "C" means "constant").

2. An affine secret bijection t from K^{37} to K^{37}. Equivalently, this parameter can be described by the 37×37 square matrix and the 37×1 column matrix over K of the transformation s with respect to the canonical basis of K^{37}. We denote by S_L the square matrix ("L" means "linear") and S_C the column matrix (here "C" means "constant").

3. A 80-bit secret string denoted by Δ.

3.2 Public Parameters

The public key consists in the function G from K^{37} to K^{26} defined by:

$$G(X) = \Big[t\Big(\varphi^{-1}\big(F(\varphi(s(X)))\big)\Big)\Big]_{0 \to 181}.$$

Here F is the function from $\mathcal{L}$ to $\mathcal{L}$ defined by:

$$\forall A \in \mathcal{L},\ F(A) = A^{128^{11}+1}.$$

By construction of the algorithm, G is a quadratic transformation over K, *i.e.* $(Y_0, \ldots, Y_{25}) = G(X_0, \ldots, X_{36})$ can be written, equivalently:

$$\begin{cases} Y_0 = P_0(X_0, \ldots, X_{36}) \\ \quad \vdots \\ Y_{25} = P_{25}(X_0, \ldots, X_{36}) \end{cases}$$

with each P_i being a quadratic polynomial of the form

$$P_i(X_0, \ldots, X_{36}) = \sum_{0 \leq j < k < 37} \zeta_{i,j,k} X_j X_k + \sum_{0 \leq j < 37} \nu_{i,j} X_j + \rho_i,$$

all the elements $\zeta_{i,j,k}$, $\nu_{i,j}$ and ρ_i being in K.

4 Generation of the key

In the SFLASH scheme, the public is deduced from the secret key, as explained in section 3.2. We need only to describe how the secret key is generated. As described in section 3.1, the following secret elements have to be generated:

- The secret invertible 37×37 matrix S_L, and the secret 37×1 (column) matrix S_C, all the coefficients being in K.
- The secret invertible 37×37 matrix T_L, and the secret 37×1 (column) matrix T_C, all the coefficients being in K.
- The 80-bit secret string Δ.

Note that, through the π transformation, generating an element of K is equivalent to generating a 7-bit string. In what follows, we call

`next_7bit_random_string`

the string of 7 bits obtained by calling 7 times the CSPRBG (we obtain first the first bit of the string, then the second bit, ..., until the seventh bit).
To generate all these parameters, we apply the following method, which uses a cryptographically secure pseudorandom bit generator (CSPRBG). From a seed whose entropy is at least 80 bits, this CSPRBG is supposed to produce a new random bit each time it is asked to.

1. To generate the invertible 37×37 matrix S_L, two methods can be used:

   First Method ("Trial and error"): Generate the matrix S_L by repeating

   ```
   for i=0 to 36
   for j=0 to 36
   S_L[i,j]=pi(next_7bit_random_string)
   ```

   until we obtain an invertible matrix.

Second Method (with the LU decomposition): Generate a lower triangular 37×37 matrix L_S and an upper triangular 37×37 matrix U_S, all the coefficients being in K, as follows:

```
for i=0 to 36
for j=0 to 36
{
if (i<j) then {U_S[i,j]=pi(next_7bit_random_string); L_S[i,j]=0}
if (i>j) then {L_S[i,j]=pi(next_7bit_random_string); U_S[i,j]=0}
if (i=j) then {repeat (z=next_7bit_random_string) until z!=(0,0,0,0,0,0,0)
               U_S[i,j]=pi(z); L_S[i,j]=1}
}
```

Define then $S_L = L_S \times U_S$.

2. Generate S_C by using the CSPRBG to obtain 37 new random elements of K (from the top to the bottom of the column matrix). Each of these elements of K is obtained by

```
pi(next_7bit_random_string)
```

3. To generate the invertible 37×37 matrix S_L, two methods can be used:

First Method ("Trial and error"): Generate the matrix T_L by repeating

```
for i=0 to 36
for j=0 to 36
T_L[i,j]=pi(next_7bit_random_string)
```

until we obtain an invertible matrix.

Second Method (with the LU decomposition): Generate a lower triangular 37×37 matrix L_T and an upper triangular 37×37 matrix U_T, all the coefficients being in K, as follows:

```
for i=0 to 36
for j=0 to 36
{
if (i<j) then {U_T[i,j]=pi(next_7bit_random_string); L_T[i,j]=0}
if (i>j) then {L_T[i,j]=pi(next_7bit_random_string); U_T[i,j]=0}
if (i=j) then {repeat (z=next_7bit_random_string) until z!=(0,0,0,0,0,0,0)
               U_T[i,j]=pi(z); L_T[i,j]=1}
}
```

Define then $T_L = L_T \times U_T$.

4. Generate T_C by using the CSPRBG to obtain 37 new random random elements of K (from the top to the bottom of the column matrix). Each of these elements of K is obtained by

```
pi(next_7bit_random_string)
```

5. Finally, generate Δ by using the CSPRBG to obtain 80 random bits.

Note that the generation of a complete secret key thus requires 20282 bits from the CSPRBG (with the second method).

5 Signing a message

In the present section, we describe the signature of a message M by the SFLASH algorithm.

5.1 The signing algorithm

The message M is given by a string of bits. Its signature S is obtained by applying successively the following operations (see figure 1):

1. Let M_1 and M_2 be the three 160-bit strings defined by:

$$M_1 = \text{SHA-1}(M),$$

$$M_2 = \text{SHA-1}(M_1).$$

2. Let V be the 182-bit string defined by:

$$V = [M_1]_{0\to 159}||[M_2]_{0\to 21}.$$

3. Let W be the 77-bit string defined by:

$$W = [\text{SHA-1}(V||\Delta)]_{0\to 76}.$$

4. Let Y be the string of 26 elements of K defined by:

$$Y = \Big(\pi([V]_{0\to 6}), \pi([V]_{7\to 13}), \ldots, \pi([V]_{175\to 181})\Big).$$

5. Let R be the string of 11 elements of K defined by:

$$R = \Big(\pi([W]_{0\to 6}), \pi([W]_{7\to 13}), \ldots, \pi([W]_{70\to 76})\Big).$$

6. Let B be the element of $\mathcal{L}$ defined by:

$$B = \varphi\Big(t^{-1}(Y||R)\Big).$$

7. Let A be the element of $\mathcal{L}$ defined by:

$$A = F^{-1}(B),$$

   F being the function from $\mathcal{L}$ to $\mathcal{L}$ defined by:

$$\forall A \in \mathcal{L},\ F(A) = A^{128^{11}+1}.$$

8. Let $X = (X_0, \ldots, X_{36})$ be the string of 37 elements of K defined by:

$$X = (X_0, \ldots, X_{36}) = s^{-1}\Big(\varphi^{-1}(A)\Big).$$

9. The signature S is the 259-bit string given by:

$$S = \pi^{-1}(X_0)|| \ldots ||\pi^{-1}(X_{36}).$$

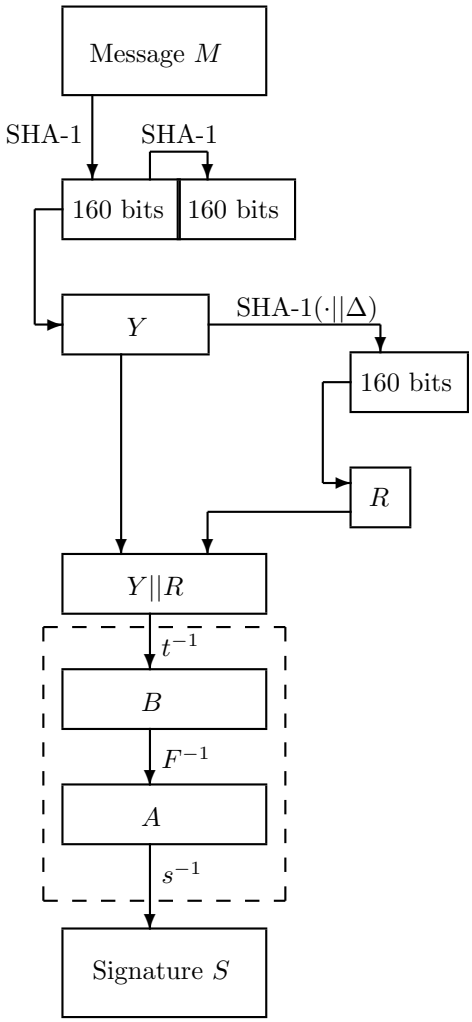

Figure 1: Signature generation with SFLASH

5.2 Computing $A = F^{-1}(B)$

The function F, from $\mathcal{L}$ to $\mathcal{L}$, is defined by:

$$\forall A \in \mathcal{L},\ F(A) = A^{128^{11}+1}.$$

As a consequence, $A = F^{-1}(B)$ can be obtained by the following formula:

$$A = B^h,$$

the value of the exponent h being the inverse of $128^{11} + 1$ modulo $128^{37} - 1$. In fact, h can be explicitly given by:

$$h = 2^{258} + \sum_{i=0}^{17} \sum_{j=154i+76}^{154i+152} 2^j.$$

Three methods can be used to compute $A = B^h$:

1. Directly compute the exponentiation B^h by using the "square-and-multiply" principle.
2. Use the following algorithm:
   (a) Initialize A to:
   $$A = B^{2^{76}} \quad \left(= B^{128^{10}} \cdot B^{64} \right).$$
   Note that $B \mapsto B^{128^{10}}$ is a linear transformation of $\mathcal{L}$ if we consider $\mathcal{L}$ as a vector space over K and can thus be easily computed.
   (b) Compute
   $$u = A^{128^{11}-1}.$$
   This value can be computed either by using the "square-and-multiply" principle or by noticing that we also have
   $$u \cdot A = A^{128^{11}}$$
   with $A \mapsto A^{128^{11}}$ being a linear transformation of $\mathcal{L}$ if we consider $\mathcal{L}$ as a vector space over K. We can thus easily find A by solving a system of linear equations over K.
   (c) Apply 18 times the following transformation: replace A by $u \cdot A^{128^{22}}$. This is also practical, since $A \mapsto A^{128^{22}}$ is a linear transformation of $\mathcal{L}$ (considered as a vector space over K).
3. Finally, we can also use the fact that
   $$A \cdot B^{128^{11}} = A^{128^{22}} \cdot B.$$
   Since $B \mapsto B^{128^{11}}$ and $A \mapsto A^{128^{22}}$ are two linear transformations of $\mathcal{L}$ (considered as a vector space over K), A can be found by solving a system of linear equations over K.

6 Verifying a signature

Given a message M (*i.e.* a string of bits) and a signature S (a 259-bit string), the following algorithm is used to decide whether S is a valid signature of M or not:

1. Let M_1 and M_2 be the three 160-bit strings defined by:

$$M_1 = \text{SHA-1}(M),$$

$$M_2 = \text{SHA-1}(M_1).$$

2. Let V be the 182-bit string defined by:

$$V = [M_1]_{0 \to 159} || [M_2]_{0 \to 21}.$$

3. Let Y be the string of 26 elements of K defined by:

$$Y = \Big(\pi([V]_{0 \to 6}), \pi([V]_{7 \to 13}), \ldots, \pi([V]_{175 \to 181})\Big).$$

4. Let Y' be the string of 26 elements of K defined by:

$$Y' = G\Big(\pi([S]_{0 \to 6}), \pi([S]_{7 \to 13}), \ldots, \pi([S]_{252 \to 258})\Big).$$

5. - If Y equals Y', accept the signature.
   - Else reject the signature.

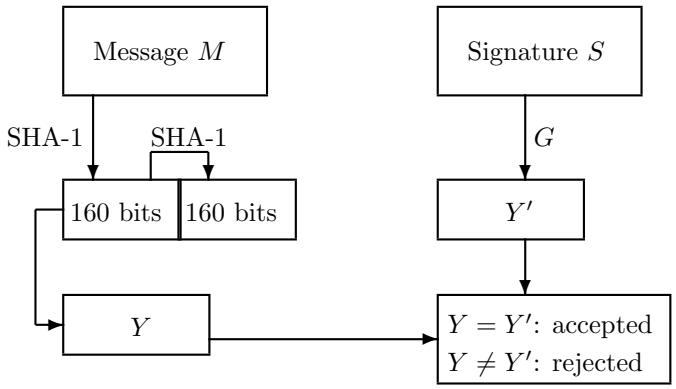

Figure 2: Signature verification with SFLASH

7 Security of the SFLASH algorithm

SFLASH is a C^{*--} scheme with a special choice of the parameters.

The security of such schemes has been studied in [2].

The security is not proven to be equivalent to a simple to describe and assumed difficult to solve problem. However, here are the present results on the two possible kinds of attacks :

7.1 Attacks that compute a valid signature from the public key as if it was a random set of quadratic equations (*i.e.* without using the fact that we have a C^{*--} scheme)

These attacks have to solve a MQ problem (MQ: Multivariate Quadratic equations), and the general MQ problem is NP-Hard. Moreover, when the parameters are well chosen, the known algorithms for solving such an MQ problem (such as XL, FXL or some Gröbner base algorithms) are efficient. With our choice of parameters for SFLASH, they require more computations than the equivalent of 2^{80} TDES operations.

7.2 Attacks that use the fact that the public key comes from a C^{*--} scheme (and is not a random set of quadratic equations)

All the known attacks on this family have a complexity in $\mathcal{O}(qr)$, where r is the number of removed equations ($r = 11$ in the SFLASH algorithm), and where q is the number of elements of the finite field K used (so $q = 128 = 2^7$ for the SFLASH algorithm). So these attacks will require more than the equivalent of 2^{80} TDES operations for the SFLASH algorithm.

8 Summary of the characteristics of SFLASH

- Length of the signature: 259 bits.
- Length of the public key: 15.4 Kbytes.
- Length of the secret key: the secret key (2.45 Kbytes) is generated from a small seed of at least 128 bits.
- Time to sign a message[1]: less than 2.7 ms (maximum time).
- Time to verify a signature[2]: less than 0.8 ms (*i.e.* approximately $37 \times 37 \times 26$ multiplications and additions in K).
- Time to generate a pair of public key/secret key: less than 1 s.
- Best known attack: more than 2^{80} TDES computations.

References

[1] W. Geiselmann, R. Steinwandt, Th. Beth, *Attacking the Affine Parts of SFLASH*, in Proceedings of the second NESSIE Workshop, 12-13 September 2001, Egham, UK.

[2] J. Patarin, L. Goubin, N. Courtois, *C^{*-+} and HM: Variations around two schemes of T. Matsumoto and H. Imai*, in Advances in Cryptology, Proceedings of ASIACRYPT'98, LNCS n° 1514, Springer Verlag, 1998, pp. 35-49.

[1] On a Pentium III 500 MHz. This part can be improved: the given software was not optimized.
[2] This part can be improved: the given software was not optimized.

9 Appendix - Changes to SFLASH.

The SFLASH signature scheme has modified, as allowed in the second stage of Nessie evaluation process. In some papers that refer to the old version, it is sometimes called SFLASH^{v1}, and SFLASH^{v2} is the new final version. The only official version of SFLASH is now SFLASH^{v2} that can be called just SFLASH.

In this section we summarize the changes, which is aimed at readers and developers that are acquainted with the previous version SFLASH^{v1}. It requires the knowledge of the previous version of SFLASH. Both in the first version of specification (SFLASH^{v1}), as well as in the main part of the present document (above) that specifies completely SFLASH^{v2}, we used the same notations.

Choosing the coefficients of s and t in GF(128) instead of GF(2)

The only modification between SFLASH^{v1} and SFLASH^{v2} consists in choosing the coefficients of the secret affine transformations s and t in the fields $K = \mathrm{GF}(128)$, instead of the subfield $K' = \mathrm{GF}(2)$.

This is done to prevent a new attack that has been very recently discovered on SFLASH^{v1} and was indicated to us by Henri Gilbert. By choosing GF(128), we combine the advantages of FLASH and SFLASH^{v1} : choosing the coefficients of s and t in the "big" field GF(128) (instead of the "small" field GF(2)) avoids the new attack; and choosing GF(128) (instead of GF(256) as in FLASH) avoids some potential other attacks due to the existence of subfields.

Due to our modification, the size of the public key has increased from 2.2 Kbytes to 15.4 Kbytes. However, this modification is essential if we want to keep a good level of security for SFLASH.

Note that with our modification, the attack described by Geiselmann, Steinwandt and Beth in their paper "Attacking the Affine Parts of SFLASH" (presented at the second NESSIE workshop, see [1]) does not apply to SFLASH^{v2}.