Quartz, an asymmetric signature scheme for short signatures on PC

Primitive specification and supporting documentation

(second revised version, October 2001)

Nicolas T. Courtois[1], Louis Goubin[1] and Jacques Patarin[2]

[1]Axalto Cryptographic Research & Advanced Security,
36-38 rue de la Princesse, BP 45, 78430 Louveciennes Cedex, France
courtois@minrank.org, LGoubin@slb.com

[2] PRISM, University of Versailles, France
Jacques.Patarin@prism.uvsq.fr

Note: This document specifies the updated final version of the Quartz signature scheme, slightly modified as allowed in the second stage of Nessie evaluation process, in order to improve the speed and the security. In some papers that refer to the old version, it is sometimes called Quartz^{v1}, and Quartz^{v2} is the new version. This is therefore the only official version of Quartz. We note that the key generation has not changed, the signature computation has changed, and the signature verification has changed slightly. In the Appendix of the present document we summarize all the changes to Quartz, for readers and developers that are acquainted with the previous version. It also includes an explanation why these changes has been made.

1 Introduction

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

Quartz is a HFEV$^-$ algorithm (see [7]) with a special choice of the parameters. Quartz 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.

Quartz is designed to generate very very short signatures: only 128 bits ! Moreover, in Quartz, all the state of the art ideas to enforce the security of such an algorithm have been used: Quartz is built on a "Basic HFE" scheme secure by itself at present (no practical attacks are known for our parameter choice) and, on this underlying scheme, we have introduced some "perturbation operations" such as removing some equations on the originally public key, and introducing some extra variables (these variables are sometime called "vinegar variables"). The resulting schemes look quite complex at first sight, but it can be seen as the resulting actions of many ideas in the same direction: to have a very short signature with maximal security (*i.e.* the "hidden" polynomial F of small degree d is hidden as well as possible).

As a result, the parameters of Quartz have been chosen in order to satisfy an extreme property that no other standardized public key scheme has reached so far: very short signatures. Quartz 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 Quartz.

Quartz 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 Notation

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

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

For a given string $\lambda = (\lambda_0, \dots, \lambda_m)$ 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:

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

3 Parameters of the algorithm

The Quartz algorithm uses the field $\mathcal{L} = \mathbf{F}_{2^{103}}$. More precisely, we chose $\mathcal{L} = \mathbf{F}_2[X]/(X^{103} + X^9 + 1)$. We will denote by φ the bijection between $\{0,1\}^{103}$ and $\mathcal{L}$ defined by:

$$\forall \omega = (\omega_0, \dots, \omega_{102}) \in \{0,1\}^{103}$$
$$\varphi(\omega) = \omega_{102} X^{102} + \ldots + \omega_1 X + \omega_0 \pmod{X^{103} + X^9 + 1}$$

3.1 Secret parameters

1. An affine secret bijection s from $\{0,1\}^{107}$ to $\{0,1\}^{107}$. Equivalently, this parameter can be described by the 107×107 square matrix and the 107×1 column matrix over $\mathbf{F}_2$ of the transformation s with respect to the canonical basis of $\{0,1\}^{107}$. 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 $\{0,1\}^{103}$ to $\{0,1\}^{103}$. Equivalently, this parameter can be described by the 103×103 square matrix and the 103×1 column matrix over $\mathbf{F}_2$ of the transformation s with respect to the canonical basis of $\{0,1\}^{103}$. We denote by T_L the square matrix ("L" means "linear") and T_C the column matrix (here "C" means "constant").

3. A family of secret functions $(F_V)_{V \in \{0,1\}^4}$ from $\mathcal{L}$ to $\mathcal{L}$, defined by:

$$F_V(Z) = \sum_{\substack{0 \leq i < j < 103 \\ 2^i + 2^j \leq 129}} \alpha_{i,j} \cdot Z^{2^i + 2^j} + \sum_{\substack{0 \leq i < 103 \\ 2^i \leq 129}} \beta_i(V) \cdot Z^{2^i} + \gamma(V).$$

In this formula, each $\alpha_{i,j}$ belongs to $\mathcal{L}$ and each β_i ($0 \le i < 103$) is an affine transformation from $\{0,1\}^4$ to $\mathcal{L}$, *i.e.* a transformation satisfying

$$\forall V = (V_0, V_1, V_2, V_3) \in \{0,1\}^4,\ \beta_i(V) = \sum_{0 \le k < 4} V_k \cdot \xi_{i,k} + \upsilon_i$$

with all the $\xi_{i,k}$ and υ_i being elements of $\mathcal{L}$. Finally, γ is a quadratic transformation from $\{0,1\}^4$ to $\mathcal{L}$, *i.e.* a transformation satisfying

$$\forall V = (V_0, V_1, V_2, V_3) \in \{0,1\}^4,\ \gamma(V) = \sum_{0 \le k < \ell < 4} V_k V_\ell \cdot \eta_{k,\ell} + \sum_{0 \le k < 4} V_k \cdot \sigma_k + \tau$$

with all the $\eta_{k,\ell}$, σ_k and τ being elements of $\mathcal{L}$.

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

3.2 Public parameters

The public key consists in the function G from $\{0,1\}^{107}$ to $\{0,1\}^{100}$ defined by:

$$G(X) = \Big[t\Big(\varphi^{-1}\big(F_{[s(X)]_{103 \to 106}}(\varphi([s(X)]_{0 \to 102}))\big)\Big)\Big]_{0 \to 99}.$$

By construction of the algorithm, G is a quadratic transformation over $\mathbf{F}_2$, *i.e.* $(Y_0, \ldots, Y_{99}) = G(X_0, \ldots, X_{106})$ can be written, equivalently:

$$\begin{cases} Y_0 = P_0(X_0, \ldots, X_{106}) \\ \quad \vdots \\ Y_{99} = P_{99}(X_0, \ldots, X_{106}) \end{cases}$$

with each P_i being a quadratic polynomial of the form

$$P_i(X_0, \ldots, X_{106}) = \sum_{0 \le j < k < 107} \zeta_{i,j,k} X_j X_k + \sum_{0 \le j < 107} \nu_{i,j} X_j + \rho_i,$$

all the elements $\zeta_{i,j,k}$, $\nu_{i,j}$ and ρ_i being in $\mathbf{F}_2$.

4 Generation of the key

In the Quartz 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 following secret elements of $\mathcal{L}$:

$$\begin{cases} \alpha_{i,j} & \text{with } 0 \le i < j < 103 \text{ and } 2^i + 2^j \le 129 \\ \xi_{i,k} & \text{with } 0 \le i < 8 \text{ and } 0 \le k < 4 \\ \upsilon_i & \text{with } 0 \le i < 8 \\ \eta_{k,\ell} & \text{with } 0 \le k < \ell < 4 \\ \sigma_k & \text{with } 0 \le k < 4 \\ \tau & \end{cases}$$

- The secret invertible 107×107 matrix S_L, and the secret 107×1 (column) matrix S_C, all the coefficients being 0 or 1.

- The secret invertible 107×107 matrix T_L, and the secret 107×1 (column) matrix T_C, all the coefficients being 0 or 1.
- The 80-bit secret string Δ.

Note that, through the φ transformation, generating an element of $\mathcal{L}$ is equivalent to generating a 103-bit string.
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. Generate the coefficients of

$$F_{(0,0,0,0)}(Z) = \sum_{\substack{0 \le i < j < 103 \\ 2^i + 2^j \le 129}} \alpha_{i,j} \cdot Z^{2^i + 2^j} + \sum_{\substack{0 \le i < 103 \\ 2^i \le 129}} \upsilon_i \cdot Z^{2^i} + \tau,$$

from the lower to the higher power of Z. More precisely, the first 103 bits produced by the CSPRBG give τ (when applying φ). We then successively generate: υ_0, υ_1, $\alpha_{0,1}$, υ_2, $\alpha_{0,2}$, $\alpha_{1,2}$, υ_3, $\alpha_{0,3}$, $\alpha_{1,3}$, $\alpha_{2,3}$, υ_4, $\alpha_{0,4}$, $\alpha_{1,4}$, $\alpha_{2,4}$, $\alpha_{3,4}$, υ_5, $\alpha_{0,5}$, $\alpha_{1,5}$, $\alpha_{2,5}$, $\alpha_{3,5}$, $\alpha_{4,5}$, υ_6, $\alpha_{0,6}$, $\alpha_{1,6}$, $\alpha_{2,6}$, $\alpha_{3,6}$, $\alpha_{4,6}$, $\alpha_{5,6}$, υ_7, $\alpha_{0,7}$ (each time, we use the CSPRBG to generate 103 new random bits, and we then apply φ).

2. Generate the coefficients of

$$F_{(1,0,0,0)}(Z) - F_{(0,0,0,0)}(Z) = \sum_{\substack{0 \le i < 103 \\ 2^i \le 129}} \xi_{i,0} \cdot Z^{2^i} + \sigma_0,$$

from the lower to the higher power of Z. More precisely, the first 103 bits produced by the CSPRBG give σ_0 (when applying φ). We then successively generate: $\xi_{0,0}$, $\xi_{1,0}$, $\xi_{2,0}$, $\xi_{3,0}$, $\xi_{4,0}$, $\xi_{5,0}$, $\xi_{6,0}$, $\xi_{7,0}$ (each time, we use the CSPRBG to generate 103 new random bits, and we then apply φ).

3. Generate the coefficients of

$$F_{(0,1,0,0)}(Z) - F_{(0,0,0,0)}(Z) = \sum_{\substack{0 \le i < 103 \\ 2^i \le 129}} \xi_{i,1} \cdot Z^{2^i} + \sigma_1,$$

from the lower to the higher power of Z. More precisely, the first 103 bits produced by the CSPRBG give σ_1 (when applying φ). We then successively generate: $\xi_{0,1}$, $\xi_{1,1}$, $\xi_{2,1}$, $\xi_{3,1}$, $\xi_{4,1}$, $\xi_{5,1}$, $\xi_{6,1}$, $\xi_{7,1}$ (each time, we use the CSPRBG to generate 103 new random bits, and we then apply φ).

4. Generate the coefficients of

$$F_{(0,0,1,0)}(Z) - F_{(0,0,0,0)}(Z) = \sum_{\substack{0 \le i < 103 \\ 2^i \le 129}} \xi_{i,2} \cdot Z^{2^i} + \sigma_2,$$

from the lower to the higher power of Z. More precisely, the first 103 bits produced by the CSPRBG give σ_2 (when applying φ). We then successively generate: $\xi_{0,2}$, $\xi_{1,2}$, $\xi_{2,2}$, $\xi_{3,2}$, $\xi_{4,2}$, $\xi_{5,2}$, $\xi_{6,2}$, $\xi_{7,2}$ (each time, we use the CSPRBG to generate 103 new random bits, and we then apply φ).

5. Generate the coefficients of

$$F_{(0,0,0,1)}(Z) - F_{(0,0,0,0)}(Z) = \sum_{\substack{0 \le i < 103 \\ 2^i \le 129}} \xi_{i,3} \cdot Z^{2^i} + \sigma_3,$$

from the lower to the higher power of Z. More precisely, the first 103 bits produced by the CSPRBG give σ_3 (when applying φ). We then successively generate: $\xi_{0,3}$, $\xi_{1,3}$, $\xi_{2,3}$, $\xi_{3,3}$, $\xi_{4,3}$, $\xi_{5,3}$, $\xi_{6,3}$, $\xi_{7,3}$ (each time, we use the CSPRBG to generate 103 new random bits, and we then apply φ).

6. Successively generate the remaining coefficients (corresponding to the quadratic part of $\gamma(V)$), in lexicographic order: $\eta_{0,1}$, $\eta_{0,2}$, $\eta_{0,3}$, $\eta_{1,2}$, $\eta_{1,3}$, $\eta_{2,3}$ (each time, we use the CSPRBG to generate 103 new random bits, and we then apply φ).

7. To generate the invertible 107×107 matrix S_L, two methods can be used:

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

```
for i=0 to 106
    for j=0 to 106
        S_L[i,j]=next_random_bit
```

   until we obtain an invertible matrix.

   Second Method (with the LU decomposition): Generate a lower triangular 107×107 matrix L_S and an upper triangular 107×107 matrix U_S, all the coefficients being 0 or 1, as follows:

```
for i=0 to 106
    for j=0 to 106
    {
        if (i<j) then {U_S[i,j]=next_random_bit; L_S[i,j]=0}
        if (i>j) then {L_S[i,j]=next_random_bit; U_S[i,j]=0}
        if (i=j) then {U_S[i,j]=1; L_S[i,j]=1}
    }
```

   Define then $S_L = L_S \times U_S$.

8. Generate S_C by using the CSPRBG to obtain 107 new random bits (from the top to the bottom of the column matrix).

9. To generate the invertible 107×107 matrix T_L, two methods can be used:

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

```
for i=0 to 106
    for j=0 to 106
        T_L[i,j]=next_random_bit
```

   until we obtain an invertible matrix.

Second Method (with the LU decomposition): Generate a lower triangular 107×107 matrix L_T and an upper triangular 107×107 matrix U_T, all the coefficients being 0 or 1, as follows:

```
for i=0 to 106
    for j=0 to 106
    {
        if (i<j) then {U_T[i,j]=next_random_bit; L_T[i,j]=0}
        if (i>j) then {L_T[i,j]=next_random_bit; U_T[i,j]=0}
        if (i=j) then {U_T[i,j]=1; L_T[i,j]=1}
    }
```

Define then $T_L = L_T \times U_T$.

10. Generate T_C by using the CSPRBG to obtain 107 new random bits (from the top to the bottom of the column matrix).

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

Note that the generation of a complete secret key thus requires 30497 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 Quartz 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_0, M_1, M_2 and M_3 be the three 160-bit strings defined by:

$$M_0 = \text{SHA-1}(M),$$

$$M_1 = \text{SHA-1}(M_0||0x00),$$
$$M_2 = \text{SHA-1}(M_0||0x01).$$
$$M_3 = \text{SHA-1}(M_0||0x02).$$

With 0x00 through 0x02 denoting one single 8-bit character appended to M_0.

2. Let H_1, H_2, H_3 and H_4 be the four 100-bit strings defined by:

$$H_1 = [M_1]_{0\to 99},$$

$$H_2 = [M_1]_{100\to 159}||[M_2]_{0\to 39},$$
$$H_3 = [M_2]_{40\to 139},$$
$$H_4 = [M_2]_{140\to 159}||[M_3]_{0\to 79}.$$

3. Let $\tilde{S}$ be a 100-bit string. $\tilde{S}$ is initialized to $00\ldots0$.

4. For $i = 1$ to 4, do

(a) Let Y be the 100-bit string defined by:

$$Y = H_i \oplus \tilde{S}.$$

(b) Let W be the 160-bit string defined by:

$$W = \text{SHA-1}(Y||\Delta).$$

(c) Let R be the 3-bit string defined by:

$$R = [W]_{0\to 2}.$$

(d) Let V be the 4-bit string defined by:

$$V = [W]_{3\to 6}.$$

(e) Let B be the element of $\mathcal{L}$ defined by:

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

(f) Consider the following univariate polynomial equation in Z (over $\mathcal{L}$):

$$F_V(Z) = B.$$

(g) If this equation $F_V(Z) = B$ has no solutions, replace W by SHA-1(W) and go back to step (c).

(h) Now the equation $F_V(Z) = B$ has one or several solutions in $\mathcal{L}$, then let $A(1), A(2), .., A(\delta)$ be these solutions.

(i) If there is only one solution, we put $A = A(1)$. Otherwise, we hash each solution $I(i) = \text{SHA-1}(A(i))$. Let A be the one that gives the smallest hash $I(i)$ in the big-endian ordering: we compare the first character in memory, then the second etc..

(j) Let X be the 107-bit string defined by:

$$X = s^{-1}\Big(\varphi^{-1}(A)||V\Big).$$

(k) Define the new value of the 100-bit string $\tilde{S}$ by:

$$\tilde{S} = [X]_{0\to 99} \ ;$$

(l) Let X_i be the 7-bit string defined by:

$$X_i = [X]_{100\to 106}.$$

5. The signature S is the 128-bit string given by:

$$S = \tilde{S}||X_4||X_3||X_2||X_1.$$

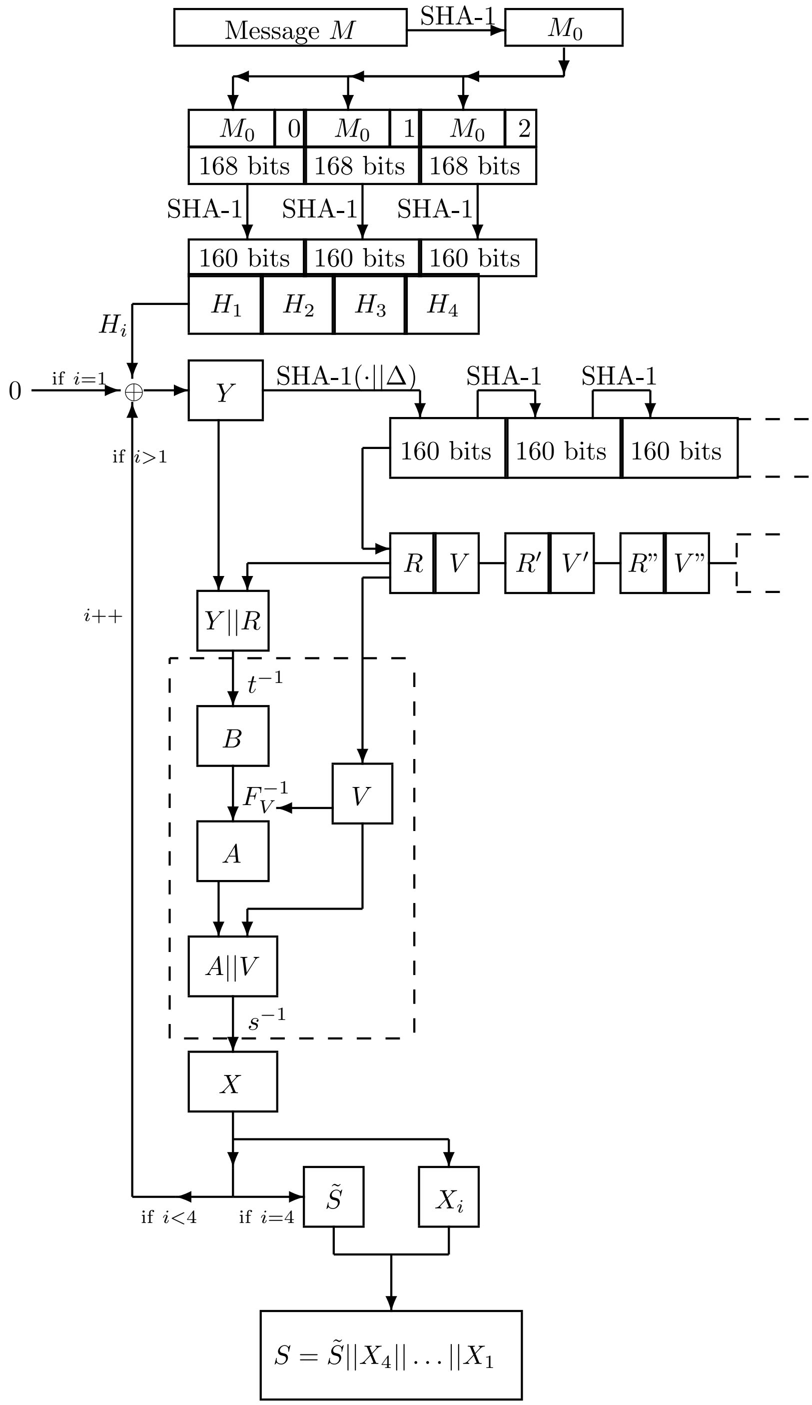

Figure 1: Signature generation with Quartz (beginning with $i = 1$)

5.2 Solving the equation $F_V(Z) = B$

To sign a message, we need to solve an equation of the form $F_V(Z) = B$, with $B \in \mathcal{L}$ and Z being the unknown, also being in $\mathcal{L}$. Basically, it has been done to satisfy the two following requirements:

- The solution should be chosen in a deterministic way in y.
- If there are several solutions, the choice is pseudo-random and depends on the secret key Δ.

On the implementation side, the first step to find roots of the polynomial is to compute:

$$\Psi(Z) = \gcd\Big(F_V(Z) - B, Z^{2^{103}} - Z\Big).$$

To compute the gcd above, we can first recursively compute $Z^{2^i} \mod (F_V(Z) - B)$ for $i = 0, 1, \ldots, 103$ and then compute $\Theta(Z) = Z^{2^{103}} - Z \mod (F_V(Z) - B)$. Finally $\Psi(Z)$ is easily obtained by

$$\Psi(Z) = \gcd\Big(F_V(Z) - B, \Theta(Z)\Big).$$

With this method, the degrees of the polynomials involved in the computation never exceed $2 \times 129 = 258$.
Note that more refined methods have also been developed to compute $\Psi(Z)$ (see [5]).

Now the equation $F_V(Z) = B$ has a number of solutions (in $\mathcal{L}$) equal to the degree of Ψ over $\mathcal{L}$. If the degree is 0, no solutions, we try again as specified in the signature algorithm (go back to step 4c).

Eventually we end up with an equation $F_V(Z) = B$ that has solutions. Then if Ψ is of degree one, it is of the form $\Psi(Z) = \kappa \cdot (Z - A)$ (with $\kappa \in \mathcal{L}$) and A is the unique solution of the equation $F_V(Z) = B$.
On the contrary, if Ψ is of degree greater than one, we need to apply some of the known algorithms to factor polynomials over finite fields, for example the Berlekamp algorithm, in order to find all the roots. We choose one of the roots in a pseudo-random deterministic way, as specified in the signature algorithm above.

5.3 Existence of the signature

The success of the signing algorithm relies on the following fact: for at least one of the successive values of the pair (R, V), there exist a solution (in Z) for the equation $F_V(Z) = B$.

It can be proven that, for a randomly chosen B, the probability of having a solution in Z is approximately $1 - \frac{1}{e}$. If we suppose that the successive values (R, V) take all the possible values in $\{0, 1\}^7$, the probability of never having a solution is approximately given by:

$$\left(\frac{1}{e}\right)^{128} \simeq 2^{-185}.$$

Since the signing algorithm has to solve this equation four times, the probability that the algorithm fails is about:

$$\mathcal{P} \simeq 1 - \left(1 - \left(\frac{1}{e}\right)^{128}\right)^4 \simeq 2^{-183}.$$

This probability is thus completely negligible.

6 Verifying a signature

6.1 The verification algorithm

Given a message M (*i.e.* a string of bits) and a signature S (a 128-bit string), the following algorithm is used to decide whether S is a valid signature of M or not:

1. Let M_0, M_1, M_2 and M_3 be the three 160-bit strings defined by:

$$M_0 = \text{SHA-1}(M),$$

$$M_1 = \text{SHA-1}(M_0||0x00),$$
$$M_2 = \text{SHA-1}(M_0||0x01).$$
$$M_3 = \text{SHA-1}(M_0||0x02).$$

   With 0x00 through 0x02 denoting one single 8-bit character appended to M_0.

2. Let H_1, H_2, H_3 and H_4 be the four 100-bit strings defined by:

$$H_1 = [M_1]_{0\to 99},$$

$$H_2 = [M_1]_{100\to 159}||[M_2]_{0\to 39},$$
$$H_3 = [M_2]_{40\to 139},$$
$$H_4 = [M_2]_{140\to 159}||[M_3]_{0\to 79}.$$

3. Let $\tilde{S}$ be the 100-bit string defined by:

$$\tilde{S} = [S]_{0\to 99}.$$

4. Let X_4, X_3, X_2, X_1 be the four 7-bit strings defined by:

$$X_4 = [S]_{100\to 106},$$

$$X_3 = [S]_{107\to 113},$$
$$X_2 = [S]_{114\to 120},$$
$$X_1 = [S]_{121\to 127}.$$

5. Let U be a 100-bit string. U is initialized to $\tilde{S}$.

6. For $i = 4$ down to 1, do

   (a) Let Y be the 100-bit string defined by:

$$Y = G(U||X_i).$$

   (b) Define the new value of the 100-bit string U by:

$$U = Y \oplus H_i.$$

7.
   - If U is equal to the 100-bit string $00\ldots 0$, accept the signature.
   - Else reject the signature.

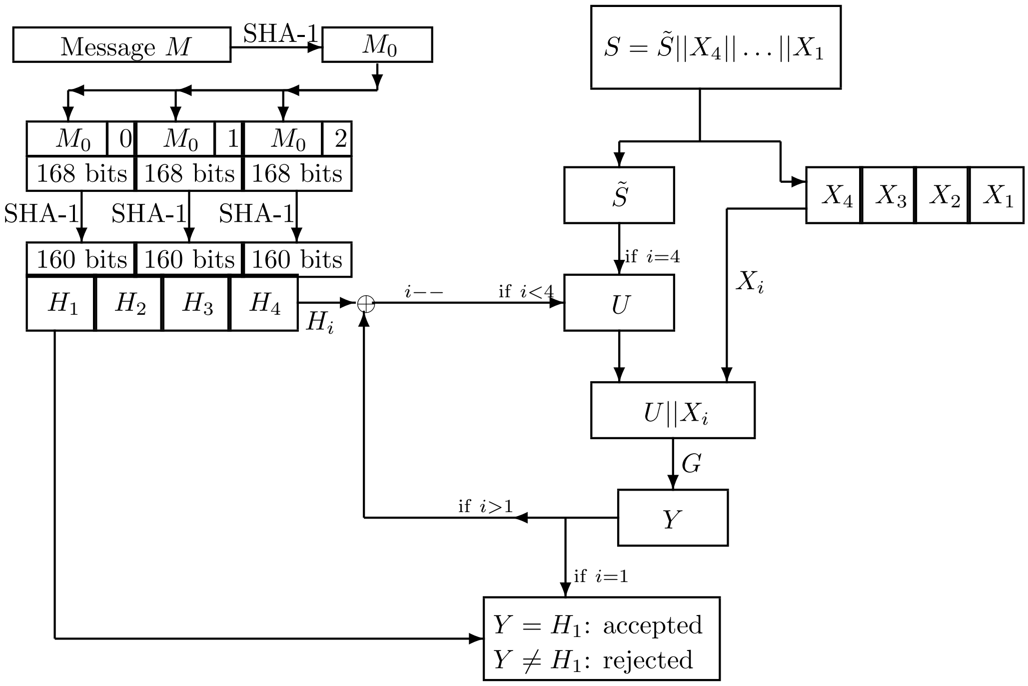

Figure 2: Signature verification with Quartz (beginning with $i = 4$)

6.2 Computation of the G function

The verification algorithm of Quartz requires the fast evaluation of the function G, which can be viewed as a set of 100 public quadratic polynomials of the form

$$P_i(x_0, \ldots, x_{106}) = \sum_{0 \leq j < k < 107} \zeta_{i,j,k} x_j x_k + \sum_{0 \leq j < 107} \nu_{i,j} x_j + \rho_i \qquad (0 \leq i \leq 99)$$

(see section 3.2).

To perform this computation, three methods can be used:

First method:

We can proceed directly, *i.e.* by successively compute the multiplications and the additions involved in P_i.

Second method:

Each of the P_i can be rewritten as follows:

$$P_i(x_0, \ldots, x_{106}) = x_0 \ell_{i,0}(x_0, \ldots, x_{106}) + x_1 \ell_{i,1}(x_1, \ldots, x_{106}) + \ldots + x_{106} \ell_{i,106}(x_{106}) + \rho_i,$$

with the $\ell_{i,0}, \ldots, \ell_{i,106}$ $(0 \leq i \leq 99)$ being 107×100 linear forms that can be explicited. As a result, since each x_j equals 0 or 1, we just have to compute modulo 2 additions of x_j variables.

Third method:

Another possible technique consists in writing

$$G(x_0, \ldots, x_{106}) = \sum_{0 \leq j < k < 107} x_j x_k \cdot Z_{j,k} \oplus \sum_{0 \leq j < 107} x_j \cdot N_j \oplus R$$

with

$$Z_{j,k} = (\zeta_{0,j,k}, \zeta_{1,j,k}, \ldots, \zeta_{99,j,k}),$$
$$N_j = (\nu_{0,j}, \nu_{1,j}, \ldots, \nu_{99,j})$$

and

$$R = (\rho_0, \rho_1, \ldots, \rho_{99}).$$

The computation can then be performed as follows:

1. Let Y be a variable in $\{0,1\}^{100}$. Let Y be initialized to $R = (\rho_0, \rho_1, \ldots, \rho_{99})$.
2. For each monomial $x_j x_k$ ($0 \leq j < k < 107$): if $x_j = x_k = 1$ then replace Y by $Y \oplus Z_{j,k}$.
3. For each monomial x_j ($0 \leq j < 107$): if $x_j = 1$ then replace Y by $Y \oplus N_j$.

If, for instance, we use a 32-bit architecture, this leads to a speed-up of the algorithm: each vector $Z_{j,k}$ or N_j or R can be stored in four 32-bit registers. By using the 32-bit XOR operation, the $\oplus$ operations can be performed 32 bits by 32 bits. This means that we compute 32 public equations simultaneously.

7 Security of the Quartz algorithm

Traditionally, the security of public key algorithms relies on a problem which is both simple to describe and has the reputation to be difficult to solve (such as the factorization problem, or the discrete logarithm problem). On the opposite, traditionally, the security of secret key algorithms and of hash functions relies (not on such a problem but) on specific arguments about the construction (such as the soundness of the Feistel construction for example) and on the fact that the known cryptanalytic tools are far to break the scheme.

There are some exceptions. For example the public key scheme based on error correcting codes (such as the McEliece scheme, or the Niedereiter scheme) or the NTRU scheme do not have a security that provably relies on a well defined problem, and some hash functions have been designed on the discrete logarithm problem.

The security of the Quartz algorithm is also not proved to be equivalent to a well defined problem. However we have a reasonable confidence in its security due to some arguments that we will present in the sections below, and these arguments are not only subjective arguments.

Remark: As an example, let $\mathcal{F}$ be the composition the five AES finalists, with five independent keys of 128 bits. Almost everybody in the cryptographic community thinks that this $\mathcal{F}$ function will be a very secure function for the next 20 years, despite the fact that it security is not provably relied on a clearly, famous, and simple to describe problem.

Our (reasonable) confidence in the security of Quartz comes from the following five different kinds of arguments, that we will explain in more details below:

1. All the known attacks are far from being efficient.
2. There is a kind of "double" security in the design of the scheme: algebraic and combinatorial.
3. MQ looks really difficult in average (not only in worst case).

4. When the degree d (of the hidden polynomial F) increases, the trapdoor progressively disappears so that all the attacks must become more and more intractable.

5. The secret key is rather long (but it can be generated from a small seed of 80 bits for example), even for computing very short signatures.

7.1 All the known attacks are far from being efficient

Three kinds of attacks have been studied so far on schemes like the basic HFE or HFEV$^-$ (Quartz is a HFEV$^-$ scheme with a special choice for the parameters).

7.2 Some attacks are designed to recover the secret key (or an equivalent information)

In this family of attack, we have the exhaustive search of the key (of course intractable) and the (much more clever) Shamir-Kipnis on the basic HFE scheme (cf [6]). However this Shamir-Kipnis attack would not be efficient on the Quartz algorithm (much more than 2^{80} computations are required) even if we removed the $-$ and V perturbations. Moreover, the Shamir-Kipnis seems to work only for the basic HFE scheme (*i.e.* without the perturbations $-$ and V) and in Quartz we have some $-$ and V. So in fact, at present for a scheme like Quartz we do not see how the Shamir-Kipnis attack may work at all.

7.3 Some attacks are designed to compute a signature S from a message M directly from the equations of the public key, as if there was no trapdoor (*i.e.* by solving a general system of quadratic equations)

The MQ (= Multivariate Quadratic) problem of solving a general set of multivariate quadratic equations is a NP-Hard problem. Some (non polynomial but sometimes better than exhaustive search) algorithms have been designed for this problem, such as some Gröbner bases algorithms, or the XL and FXL algorithms (see [1]) but for our choices of the Quartz parameters, all these algorithms need more than 2^{80} computations.

7.4 Some attacks are designed to compute a signature S from a message M by detecting some difference on the public key compared to a system of general quadratic equations

Many analysis have been made in these lines of attacks. Some "affine multiple attacks" have been design, and many variations around these attacks ("higher degree attacks" etc). At present, with the parameters of the Quartz algorithm all these attacks need more the 2^{80} computations.

7.5 There is a kind of "double layered" security in the design of the scheme: algebraic and combinatorial

The security of the basic HFE scheme (*i.e.* a HFE scheme with no perturbations such as $-$ and V) can be considered as a kind of "Algebraic" problem. This is because of the Shamir-Kipnis attack that reduces HFE to the MinRank problem on very large algebraic fields, see [2, 6]. The general MinRank problem is NP-Hard, and even if for the basic HFE, the MinRank instances may not be NP-Hard, the best attacks known on MinRank problem that is obtained from HFE are still not polynomial, as long as the HFE degree d is not fixed with $d = \mathcal{O}(n)$ for example, see [2, 4].

The basic HFE scheme is Hidden in the Quartz algorithm with the perturbations $-$ and V and the above attacks does not apply to Quartz. To remove these perturbations seems to be a very difficult combinatorial problem. In order to break the Quartz scheme, it is expected that a cryptanalyst will have to solve a double problem: Combinatorial and Algebraic, and these problems do not appear separately but in a deeply mixed way, in the public key.

7.6 MQ looks really difficult in average (not only in worst case)

In the past, some public key schemes apparently (not provably) based on some NP-Hard problems, such as the Knapsack problem were broken. However the MQ problem (*i.e.* solving a general set of multivariate quadratic equations) seems to be a much more difficult problem to solve than the Knapsack Problem: on the Knapsack Problem an algorithm such as LLL is very often efficient, while on the opposite, for the MQ problem, all the known algorithms are not significantly better than exhaustive search when the number m of equations is about the same as the number n of variables and is larger than about 12.

It is also interesting to notice that almost all the "Knapsack Schemes" were broken due to a generic algorithm on the general Knapsack problem (algorithm LLL) and not with a specific attack on the trapdoor hidden in a general knapsack instance. Something similar seems to happen with the schemes based on error correcting codes, such as the McEliece or Niederreiter schemes: so far the best attacks on these schemes try to solve the general (and NP-Hard) problem of finding a word of small weight in a general linear or affine space, and are still unable to use the fact that the security of a specific trapdoor instance is probably not equivalent to solving the general problem. Currently for Quartz, as for all the mentioned schemes, the structural attacks are behind the generic attacks on the base problem. Then for Quartz the basic problem is the MQ problem that looks really very difficult.

7.7 When the degree d (of the hidden polynomial F) increases, the trapdoor progressively disappears so that all the attacks must become more and more intractable

The degree d of the Quartz algorithm is fixed to 129. However if d was not fixed, and d could be close to 2^h ($h = 103$ in the Quartz algorithm), then all the possible systems of quadratic equations could appear as the public key, so the problem of solving it would be exactly as hard as the general MQ problem (on this number of variables). Of course, we have fixed d to 129 in order to be able to compute a signature in a reasonable time on a computer, but this result shows that when d increases, the trapdoor progressively disappears, so that all the attacks must become more and more intractable. So d is really an important "security parameter". Our choice of $d = 129$ has been made to be far from the current state of the art on the cryptanalysis with small d, while still having a reasonable time on a computer to compute a signature.

7.8 The secret key is rather long (but it can be generated from a small seed of 80 bits for example), even for computing very short signatures

Many secrets are used in Quartz: the secret affine permutations s and s, the secret function F, the secret vinegar variables V, and the secret removed equations. To specify all the secret we need a rather long secret key. However, it is also possible to compute this secret key from a short seed by using any pseudorandom bit generator. In general the time to generate the secrets from the small seed will not increase a lot the time to generate a signature. Moreover it has to be done only once if we can store the secret key in a safe way on a computer or a smart card. So for practical applications it is always possible to generate the secret key from a seed of, say, 80 bits, but this secret key for a cryptanalyst of Quartz will always be similar to a much larger secret key.

So Quartz has a property that already existed in schemes like DSS (where the lengths of p and q are different): the length of the secret key is not directly linked to the length of the signature. (This property does not exist in RSA, where the length of the secret key is never larger than the length of the signature. It explains why a Quartz or DSS signature can be much smaller than a RSA signature).

The fact that a cryptanalyst of Quartz has to face such a large secret key, may also be an argument to say that in practice the time to find a Quartz secret key may be intractable in practice, even if a new sub-exponential algorithm is found and used. (So far many cryptanalyses, such as the "affine multiple attacks", have to solve huge systems of linear equations by Gaussian reductions, and often the number of variables in these systems increases very quickly with the length of the secret, so these attacks become impractical due to space and time limitations). However this argument is not as convincing, and maybe not as strong, as the other arguments presented above.

8 Summary of the characteristics of Quartz

- Length of the signature: 128 bits.
- Length of the public key: 71 Kbytes.
- Length of the secret key: the secret key (3 Kbytes) is generated from a small seed of at least 128 bits.
- Time to generate the public key[1]: 4 seconds.
- Time to sign a message[1]: 10 seconds on average.
- Time to verify a signature[1,2]: less than 1 ms.
- Best known attack: more than 2^{80} TDES computations.

[1]On a Pentium II 500 MHz.
[2]For a short message of less than 512 bits.

References

[1] N. Courtois, A. Shamir, J. Patarin, A. Klimov, *Efficient Algorithms for solving Overdefined Systems of Multivariate Polynomial Equations*, in Advances in Cryptology, Proceedings of EUROCRYPT'2000, LNCS n° 1807, Springer, 2000, pp. 392-407.

[2] Nicolas Courtois: *The security of Hidden Field Equations (HFE)*; Cryptographers' Track RSA Conference 2001, LNCS 2020, Springer, pp. 266-281.

[3] Nicolas Courtois: *Generic Attacks and the Security of Quartz,* PKC 2003, , LNCS 2567, pp. 351-364. A preliminary version was presented at the second Nessie workshop, Royal Holloway, University of London, September 2001.

[4] Nicolas Courtois: *The security of cryptographic primitives based on multivariate algebraic problems: MQ, MinRank, IP, HFE*; PhD thesis, September 25th 2001, Paris 6 University, France. Mostly in French. Available at `http://www.minrank.org/phd.pdf`

[5] E. Kaltofen, V. Shoup, *Fast polynomial factorization over high algebraic extensions of finite fields*, in Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, 1997.

[6] A. Kipnis, A. Shamir, *Cryptanalysis of the HFE public key cryptosystem*, in Advances in Cryptology, Proceedings of Crypto'99, LNCS n° 1666, Springer, 1999, pp. 19-30.

[7] J. Patarin, *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms*, in Advances in Cryptology, Proceedings of EUROCRYPT'96, LNCS n° 1070, Springer, 1996, pp. 33-48.

[8] Quartz, submitted to Nessie European call for cryptographic primitives `http://www.cryptonessie.org`. The official web page of Quartz is `http://www.minrank.org/quartz/`.

[9] The HFE cryptosystem web page: `http://www.hfe.info`

9 Appendix - Changes to Quartz.

The Quartz 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 Quartz^{v1}, and Quartz^{v2} is the new final version. The only official version of Quartz is now Quartz^{v2} that can be called just Quartz.

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

We note that the key generation has not changed, the signature computation has changed, and the signature verification has changed slightly.

9.1 Changes in message hashing

The following was done in the previous version that computes the H_i:

$$M_1 = \text{SHA-1}(M),$$
$$M_2 = \text{SHA-1}(M_1),$$
$$M_3 = \text{SHA-1}(M_2).$$

The next step was (it has not changed): to derive H_1, H_2, H_3 and H_4 as four 100-bit strings defined by:

$$H_1 = [M_1]_{0\to 99}, \qquad H_2 = [M_1]_{100\to 159}||[M_2]_{0\to 39},$$
$$H_3 = [M_2]_{40\to 139}, \qquad H_4 = [M_2]_{140\to 159}||[M_3]_{0\to 79}.$$

This cannot be considered as random, as from the first 160 bits of (H_1, H_2, H_3, H_4), one can compute the remaining 240 bits. However it is necessary for the security proofs of Quartz, see [3], that the joint distribution (H_1, H_2, H_3, H_4) behaves as a random oracle. Therefore, the above computation of the M_i has been replaced by the following:

Let M_0, M_1, M_2 and M_3 be the three 160-bit strings defined by:

$$M_0 = \text{SHA-1}(M),$$

$$M_1 = \text{SHA-1}(M_0||0x00),$$
$$M_2 = \text{SHA-1}(M_0||0x01).$$
$$M_3 = \text{SHA-1}(M_0||0x02).$$

In the above, exactly one 8-bit character is appended each time to M_0.
The H_1, H_2, H_3 and H_4 are computed from the M_i as before.

9.2 Changes in inversion of F_V

In the previous version of Quartz^{v1}, we solve in Z the following equation (step 4f as on page 7 of the present version):

$$F_V(Z) = B.$$

In the previous version we only accepted if there was exactly one solution. In the new version, we always accept if there are solutions. There remains to see which solution is chosen. For this in Quartz^{v2} we write all the solutions $A(1), A(2), .., A(\delta)$ and we hash each solution $I(i) = \text{SHA-1}(A(i))$. Let A be the $A(i)$ that gives the smallest hash

$I(i)$ in the big-endian ordering: we compare the first character in memory, then the second etc.

This change allows Quartz signatures to be about 40 % faster. This is because the probability of success for solving the above equation $F_V(Z) = B$ is now $1 - \frac{1}{e} \approx 0.63$ instead of $\frac{1}{e} \approx 0.38$ previously, and therefore we have to do about $\frac{\frac{1}{e}}{1-\frac{1}{e}} \approx 0.58$ times as much tries, i.e. about 40% less.

We also note that the new method decreases the probability that there is no signature for a given message from 2^{-83} previously, to 2^{-183}. Since all the messages are hashed on 160 bits, for most Quartz secret/public key pairs, there is no message that has no signature, and for other keys we may consider that such message will never be found.