CFS Software Implementation

Gregory Landais and Nicolas Sendrier

INRIA Paris-Rocquencourt, Project-Team SECRET
{gregory.landais,nicolas.sendrier}@inria.fr

Abstract. CFS is the first practical code-based signature scheme. In the present paper, we present the initial scheme and its evolutions, the attacks it had to face and the countermeasures applied. We will show that all things considered the system remains practical and we present a software implementation of the signing primitive. For eighty bits of security our implementation produces a signature in 1.3 seconds on a single core of Intel Xeon W3670 at 3.20 GHz. Moreover the computation is easy to distribute and we can take full profit of multicore processors reducing the signature time to a fraction of second in software.

Keywords: CFS, digital signature scheme, software implementation

1 Introduction

CFS [1] is a digital signature scheme based on the McEliece cryptosystem [2]. It was published in 2001 and relies on the hardness of the syndrome decoding problem and on the undistinguishability of binary Goppa code.

The practicality of the scheme is questionable because of a large public key and a long signing time. The large key size might be a problem for some applications, but a storage requirement of a few megabytes on the verifier's side is not always an issue. The second drawback is the long signing time. In fact each signature requires a large number (several hundred of thousands) of algebraic decoding attempts. The first reported signing time in software [1] was about one minute but was only meant as a proof of concept. An FPGA implementation was reported in [3] with a signing time under one second. One of the purposes of this paper is to provide a more accurate measure in software.

Since it was published, the system has been weakened in several ways. It was recently proven in [4] that the public key of CFS could be distinguished in polynomial time from a random binary matrix. This property certainly needs to be investigated further, but at this time does not seem to lead to an effective attack. More threatening is the Bleichenbacher's attack (unpublished) which essentially reduces the cost of the best decoding attack from $2^{r/2}$ to $2^{r/3}$ ($r = 144$ in [1]) and necessitate an increase of parameters leading to a rather cumbersome scheme (gigabytes of key and minutes of signing time). Fortunately an efficient countermeasure, Parallel-CFS, was proposed in [5].

The purpose of this work is to (try to) clarify the situation, to show that the system is practical and secure with properly chosen parameters. We propose an efficient software implementation of the CFS scheme with a focus on the signing primitive. In particular we will give a precise report on the time consuming parts of the signing process. This could be an help to extend this work to coprocessor dedicated for signing. For now, our target platform is a standard PC.

We will first review the CFS scheme, its attacks and the Parallel-CFS variant. This will allow us to propose some sets of secure parameters. Next we will describe the software implementation. More specifically, we will explain the various algorithmic options for decoding binary Goppa and compare them. We will conclude with some timings and detailed measurements of our implementation.

2 Background

In all this paper we will consider only binary linear codes. Most of the statements are easily generalized to a larger alphabet, but no practical CFS-like signature scheme has ever been proposed so far with non binary codes.

2.1 Syndrome Decoding

We consider the following problem:

Computational Syndrome Decoding Problem: Given a matrix $H \in \{0,1\}^{r \times n}$, a word $s \in \{0,1\}^r$, and an integer $w > 0$, find $e \in \{0,1\}^n$ of Hamming weight $\leq w$ such that $He^T = s$.

We denote $\mathrm{CSD}(H, s, w)$ this problem as well as the set of its solution for a given instance. This problem is NP-hard [6]. For suitable parameters n, r and w it is conjectured difficult on average which is the basis for the security of many code-based cryptosystems.

2.2 Binary Goppa Codes

Let $\mathbf{F}_{2^m}$ denote the finite field with 2^m elements. Let $n \leq 2^m$, let the *support* $L = (\alpha_0, \ldots, \alpha_{n-1})$ be an ordered sequence of distinct elements of $\mathbf{F}_{2^m}$, and let the *generator* $g(z) \in \mathbf{F}_{2^m}$ be a monic irreducible polynomial of degree t. The binary Goppa code of support L and generator g is defined as

$$\Gamma(L, g) = \{(a_0, \ldots, a_{n-1}) \in \{0,1\}^n \mid \sum_{j=0}^{n-1} \frac{a_j}{z - \alpha_j} \bmod g(z) = 0\}.$$

This code has length $n \leq 2^m$ dimension $\geq n - mt$ and has an algebraic t-error correcting procedure. For the signature we will always take $n = 2^m$, a smaller value would increase the signing cost. For parameters of interest the dimension will always be exactly $k = n - mt$, we will denote $r = mt$ the codimension.

In a code-based cryptosystem using Goppa codes the system parameters are (m, t) and are known to everyone, the secret key is the pair (L, g) and the public key is a parity check matrix $H \in \{0,1\}^{r \times n}$.

Density of Decodable Syndromes for a Goppa Code: A syndrome $s \in \{0,1\}^r$ is decodable (relatively to H) with the algebraic decoder if and only if it is of the form $s = eH^T$ with e of Hamming weight t or less. There are $\sum_{i=0}^{t} \binom{n}{i} \approx \binom{n}{t}$ such syndromes. The total number of syndrome is 2^r and thus the proportion of syndrome decodable with the binary Goppa code algebraic decoder is close to

$$\frac{\binom{n}{t}}{2^r} = \frac{\binom{2^m}{t}}{2^{mt}} = \frac{2^m(2^m-1)\cdots(2^m-t+1)}{t!2^{mt}} \approx \frac{1}{t!}. \tag{1}$$

2.3 Complete Decoding

A complete decoder for a binary linear code defined by some parity check matrix $H \in \{0,1\}^{r\times n}$ is a procedure that will return for any syndrome $s \in \{0,1\}^r$ an error pattern e of minimal weight such that $eH^T = s$. The expected weight w of e will be the integer just above the Gilbert-Varshamov radius τ_{gv}, which we define as the real number[1] such that $\binom{n}{\tau_{\text{gv}}} = 2^r$. The threshold effect can be observed on two examples in Table 1.

In practice we will relax things a little bit and when we mention a complete decoder we mean a w-bounded decoder (with $w \geq \tau_{\text{gv}}$), that is a procedure $\psi : \{0,1\}^r \to \{0,1\}^n$ returning an error pattern matching with the input of weight $\leq w$ every time there exists one. A w-bounded decoder may return an error pattern which not of minimal weight. Also, a w-bounded decoder may fail even if $w \geq \tau_{\text{gv}}$ (see Table 1 for $(m,t) = (20,8)$ and $w = 9 > \tau_{\text{gv}} = 8.91$), in that case we may either choose a decoding bound larger than $\tau_{\text{gv}} + 1$ (loosing some security) or handle somehow the decoding failure (see §5.4). In the sequel, whenever the refer to complete decoding we implicitly define a decoding bound, an integer larger than τ_{gv}, denoted w.

(m,t)	τ_{gv}	$w=8$	$w=9$	$w=10$	$w=11$
(20,8)	8.91	$1-2^{-15}$	0.055	$2^{-131583}$	$2^{-10^{10}}$
(18,9)	10.26	$1-2^{-33}$	$1-2^{-18}$	0.93	2^{-2484}

Table 1. Failure probability of a w-bounded decoder for a code of length $n = 2^m$ and codimension $r = mt$

2.4 Original CFS

A CFS instance is defined by a binary Goppa code Γ of length n correcting up to t errors; of parity check matrix H; over the finite field $\mathbf{F}_2$. We will denote *decode* the decoding function of Γ. This function takes a binary syndrome as input and returns a tuple of t error positions matching with the input syndrome or fails if

[1] the mapping $x \mapsto \binom{n}{x}$ is easily extended continuously for the positive real numbers making the definition of τ_{gv} sound

no such error pattern exists. The matrix H is public and the procedure *decode* is secret. Signing a document is done like this :

1. Hash the document.
2. Suppose the hash value is a syndrome and try to decode it using Γ.
3. Use the resulting error pattern as a signature.

Since the hash value of the document is very unlikely to be a decodable syndrome (i.e. syndrome of a word at Hamming distance t or less from a codeword), step 2 is a little more complicated. CFS comes with two workarounds for this step :

- Complete decoding [Algorithm 1] that adds a certain amount of columns of H to the syndrome until it becomes decodable (*i.e.* guess a few errors).
- Counter appending that alters the message with a counter and rehash it until it becomes decodable.

The two methods require $t!$ decoding in average (a consequence of (1), see [1]), but the counter appending method includes the hash function inside the decoding thus forcing to implement it on the target architecture, which might be an inconvenience on a dedicated coprocessor. Moreover, with this method the size of the signature is variable because the counter has a high standard variation. Finally, the Parallel-CFS countermeasure (see §2.6) does not work with the counter appending method.

Algorithm 1 Signing with complete decoding

```
function SIGN($M, w$)                                ▷ input: message $M$; integer $w > t$
    $s \longleftarrow h(M)$
    loop
        $(i_{t+1}, \ldots, i_w) \xleftarrow{R} \{0, \ldots, n-1\}^{w-t}$
        $(i_1, \ldots, i_t) \longleftarrow \mathrm{decode}(s + H_{i_{t+1}} + \ldots + H_{i_w}, t)$
        if $(i_1, \ldots, i_t) \neq$ fail then return $(i_1, \ldots, i_w)$
        end if
    end loop
end function
```

2.5 Attacks

There exists key distinguishing attacks on CFS [4]: it is possible to efficiently distinguish a CFS public key (a parity check matrix) from a random matrix of same size. However this does not lead, for the moment, to any efficient key recovery attack. In practice, the best known techniques for forging a signature are based on generic decoding of linear codes, that is solving the computational syndrome decoding problem (CSD).

The two main techniques for solving CSD are Information Set Decoding (ISD) and the Generalized Birthday Algorithm (GBA). ISD was introduced by Prange in 1962 [7]. All practical variant derive from Stern's algorithm [8], the most recent improvements are [9, 10]. GBA was introduced by Wagner in 2002 [11], but and order 2 variant was already proposed in [12]. Its first use for decoding came later [13].

Decoding One Out of Many (DOOM): In the signature forgery domain, an attacker can create any number of messages suiting him and be satisfied with one on them being signed. The benefits of having access to several syndromes has been mentioned by Bleichenbacher for GBA[2]. For ISD a proposal was made in [15] and was later generalized and analyzed in [16]. It shows that if N target syndromes are given and if decoding anyone of them is enough, the time complexity is reduced by a factor almost $\sqrt{N}$ compared to the situation where a single specific syndrome has to be decoded. There is a upper limit for N after which there is no gain, it depends on the type of algorithm (ISD or GBA) and on the code parameters. In practice this would mean, for 80 bits of security, multiplying the key size by 400 with a similar signing time, or multiplying the key size by 100 and the signing time by 10.

2.6 Parallel-CFS: a Countermeasure to DOOM

Parallel CFS is a countermeasure proposed by M. Finiasz in 2010 [5], aiming at cancelling the benefits of having multiple target syndromes. The idea is to produce λ different hash values from the document to be signed (typically two to four) and to sign (that is decode) each of them separately. The final signature will be the collection of the signatures of all those hash values, see Algorithm 2 for the description using complete decoding. This way, if the attacker forges a signature for the first hash value of one of his multiple messages, he also has to forge a signature for the remaining hash values of this specific message, thus he is back to the initial single target decoding problem. As mentioned in [5], signing appending a counter to the message is impossible in this countermeasure since it is necessary to decode several hashes of the exact same message and the counter alters the message. This countermeasure increases by a factor λ the signature time, signature size and verification time.

Algorithm 2 Parallel-signing with complete decoding

```
function SIGN_MULT($M, w, \lambda$)        ▷ input: message $M$; integers $w > t$, $\lambda > 0$
    for $1 \leq i \leq \lambda$ do
        $s_i \longleftarrow$ SIGN($(M, i), w$)
    end for
    return $(s_i)_{1 \leq i \leq \lambda}$
end function
```

[2] this attack was presented in 2004 but was never published, it is described in [14]

In [5], Bleichenbacher's attack is generalized for attacking several hash values. The analysis shows that for most parameters, three hash value, sometimes only two, will cancel the benefits of the attack. For ISD, it is shown in [16] that the benefit of DOOM is not as high as for GBA. There was no generalization as in [5] for several hash values, but it is not likely to change the situation and if the number of hash values is large enough to cancel DOOM-GBA it will probably also cancel DOOM-ISD.

2.7 Previous Implementations

We are not aware of any publicly available software implementation of CFS. There is one FPGA implementation, described in [3], for the original parameters $n = 2^{16}$, $t = 9$, and $w = 11$. It reports an average signing time of 0.86 seconds and implements the Berlekamp-Massey decoding algorithm.

3 Parameter Selection

For single instances ISD is more efficient than GBA, and for multiple instances GBA-DOOM (i.e. generalized Bleichenbacher's attack) is more efficient than ISD-DOOM. To select secure parameters will look for parameters such that we are above the security requirements for the cost of the following attacks:

- ISD-MMT [10], the best known variant of ISD, for solving λ distinct single instances. In [10], only the asymptotic decoding exponent is given. We provide in appendix §A a non asymptotic analysis which we used for Table 2. We also mention the cost of a previous variant ISD-Dum [17] which is more flexible and may have an advantage in some cases (not here though). The numbers for ISD-Dum are derived from [18].
- GBA-DOOM [5], that is the generalized Bleichenbacher's attack, for Parallel-CFS of multiplicity λ.

					failure	public	security bits ($\log_2$ of binary ops.)		
m	t	τ_{gv}	w	λ	prob.	key size	ISD-MMT	ISD-Dum	GBA-DOOM
16	9	10.46	11	3	~ 0	1 MB	77.4	78.7	74.9
18	9	10.26	11	3	~ 0	5 MB	87.1	87.1	83.4
18	9	10.26	11	4	~ 0	5 MB	87.5	87.5	87.0
20	8	8.91	10	3	~ 0	20 MB	82.6	85.7	82.5
20	8	8.91	9	5	5.5%	20 MB	87.9	91.0	87.3
24	10	11.05	12	3	~ 0	500 MB	126.4	126.9	120.4
26	9	9.82	10	4	10^{-8}	2 GB	125.4	127.5	122.0

Table 2. Some parameter sets for Parallel-CFS using full length binary Goppa codes

The Table 2 gives the main features (including security) for some sets of parameters. The original parameters are given for reference but they are a bit undersized. We propose two main families of Goppa codes: 9-error correcting of length 2^{18} and 8-error correcting of length 2^{20}. The latter is faster but also has a larger public key size. All proposed parameter sets achieve 80 bits of security, our main targets are those where the hash multiplicity is $\lambda = 3$. We also give some sets of parameters with higher security which were not implemented.

To be thorough, there is a very recent improvement of ISD [19]. From what we understand of this variant of ISD-MMT it is not likely to provide a significant non-asymptotic improvement when the target weight is small compared with the length as it is the case for CFS signature.

4 Algebraic Decoding of Goppa Codes

The secret is a binary Goppa code $\Gamma(L, g)$ of length $n = 2^m$ of generator $g(z) \in \mathbf{F}_{2^m}[z]$, monic irreducible of degree t, and support $L = (\alpha_0, \ldots, \alpha_{n-1})$, consisting of (all) distinct elements of $\mathbf{F}_{2^m}$ in a specific order. The public key H is a systematic parity check matrix of $\Gamma(L, g)$. We denote $L_S = (\beta_0, \ldots, \beta_{r-1})$ the support elements corresponding to the identity part of H. An algebraic decoder for Goppa codes takes as input a binary syndrome $s = (s_0, \ldots, s_{r-1}) \in \{0, 1\}^r$ and returns, if it exists, an error pattern $e \in \{0, 1\}^n$ of weight t such that $eH^T = s$. There are several algorithms (described later in this section) which all have the same three steps:

1. Compute from s a new polynomial syndrome with coefficients in $\mathbf{F}_{2^m}$.
2. Solve a key equation relating this syndrome to the error locator polynomial.
3. Extract the roots of the locator polynomial to recover the error positions.

4.1 Goppa Key Equation

The algebraic syndrome $R(z) = \sum_{0 \leq j < r} s_j f_{\beta_j}(z)$ corresponding to s is computed as a sum of elementary syndromes $f_\beta(z)$ defined for any $\beta \in \mathbf{F}_{2^m}$ as

$$f_\beta(z) = \frac{1}{z - \beta} \bmod g(z) = \frac{1}{g(\beta)} \frac{g(z) - g(\beta)}{z - \beta}. \tag{2}$$

The corresponding key equation is

$$\sigma(z)R(z) = \frac{d}{dz}\sigma(z) \bmod g(z), \deg \sigma \leq t \tag{3}$$

which has a unique solution $\sigma(z) \in \mathbf{F}_{2^m}[z]$ up to a scalar multiplicative constant. If there exists an error pattern $e \in \{0, 1\}^n$ of weight $\leq t$ such that $eH^T = s$, then any solution to (3) is a scalar multiple of $\sigma(z) = \prod_{\beta \in \text{supp}(e)}(z - \beta)$ the locator polynomial of e (supp(e) is the subset of L corresponding to the non-zero coordinates of e). Equation (3) is solved with the Patterson algorithm [20].

4.2 Alternant Key Equation

A binary Goppa $\mathit{\Gamma}(L,g)$ can also be viewed as an alternant code. We use the fact that $\mathit{\Gamma}(L,g) = \mathit{\Gamma}(L,g^2)$ when g is square-free. We still have $R(z) = \sum_{0\leq j<r} s_j f_{\beta_j}(z)$ but the elementary syndrome $f_\beta(z)$ has degree $2t-1$ instead of $t-1$ and is now defined for any $\beta \in \mathbf{F}_{2^m}$ as

$$f_\beta(z) = \frac{1}{g(\beta)^2}\frac{1}{1-\beta z} \bmod z^{2t} = \sum_{i=0}^{2t-1} \frac{\beta^i z^i}{g(\beta)^2}. \quad (4)$$

The corresponding key equation is

$$\sigma_{inv}(z)R(z) = \omega(z) \bmod z^{2t}, \deg\omega < t, \deg\sigma_{inv} \leq t, \quad (5)$$

which has a unique solution $(\sigma_{inv}(z),\omega(z)) \in \mathbf{F}_{2^m}[z]^2$ up to a scalar multiplicative constant. If there exists an error pattern $e \in \{0,1\}^n$ of weight exactly t such that $eH^T = s$ and if $(\sigma_{inv}(z),\omega(z))$ is a solution to (5) then $\sigma(z) = z^t\sigma_{inv}(z^{-1}) = \prod_{\beta\in\mathrm{supp}(e)}(z-\beta)$ up to a scalar multiple. To remain consistent with the Goppa key equation we will speak of $\sigma(z) = z^t\sigma_{inv}(z^{-1})$ as the solution of the equation. The resolution of (5) is achieved either with the Berlekamp-Massey algorithm [21] or with the extended Euclidean algorithm.

4.3 Root Finding

The state of the art for root finding is the Berlekamp trace algorithm [22]. Its complexity is $O((m+t)t^2)$ and is advantageous compared with exhaustive techniques like Chien search or Horner's polynomial evaluation whose complexity is linear in the length n and thus exponential in m.

5 Implementation

5.1 Finite Field Arithmetic

We need to implement extensions of the binary field $\mathbf{F}_2$ of degree $m = 18$ and $m = 20$. For fields of small size, the best approach is to tabulate the logarithm and the exponentiation in base α, a primitive element of $\mathbf{F}_{2^m}$. This is efficient as long as the table fits into the processor cache. This is not the case here and we chose to implement those fields as an extension of degree 2 of $\mathbf{F}_{2^{m/2}}$. We used

$$\mathbf{F}_{2^{20}} = \mathbf{F}_{2^{10}}[x]/(x^2+x+\alpha), \mathbf{F}_{2^{10}} = \mathbf{F}_2[x]/(x^{10}+x^9+x^7+x^6+1)$$

with α a primitive element of $\mathbf{F}_{2^{10}}$ such that $\alpha^{10}+\alpha^9+\alpha^7+\alpha^6+1 = 0$, and

$$\mathbf{F}_{2^{18}} = \mathbf{F}_{2^9}[x]/(x^2+x+1), \mathbf{F}_{2^9} = \mathbf{F}_2[x]/(x^9+x^5+1).$$

The field $\mathbf{F}_{2^9}$ and $\mathbf{F}_{2^{10}}$ are small enough to be tabulated (with no cache miss on our target platform) and with Karatsuba's speedup a multiplication in the extension field requires three multiplications in the base field. For constrained architecture higher extension towers might be more effective. Also, bit slicing might offer an interesting alternative which, furthermore, is available also for prime extension degrees like $m = 19$ which have no subfield except $\mathbf{F}_2$.

5.2 Decoding

When signing a message M, we compute a hash value $s = h(M)$ considered as a syndrome according to the public key H. The word e of minimal weight such that $s = eH^T$ has weight $w > t$ and thus s cannot be decoded with the algebraic decoder which is limited to t errors. If, as described in Algorithm 1, we correctly guess $\delta = w - t$ error positions we will be able to successfully apply the algebraic decoder on a modified syndrome. It was proven in [1] that this succeeds on average after $t!$ guesses. We describe in Algorithm 3 a variant where the syndrome is modified in polynomial form. Also, the complete root finding procedure is applied once only.

Algorithm 3 Signing with binary Goppa codes

```
function SIGN($M$)                                        ▷ input: message $M$
    $s \longleftarrow h(M)$
    $R_0(z) \longleftarrow \sum_{0 \leq j < r} s_j f_{\beta_j}(z)$                  ▷ once only, either (2) or (4)
    for all $B \subset L$ of cardinality $\delta = w - t$ do
        $R(z) \longleftarrow R_0(z) + \sum_{\beta \in B} f_\beta(z)$      ▷ syndrome adjustment, either (2) or (4)
        $\sigma(z) \longleftarrow$ solve_key_eq($R(z)$)          ▷ key equation solving, either (3) or (5)
        if $z^{2^m} = z \bmod \sigma(z)$ then                          ▷ split checking
            $A \longleftarrow$ roots_of($\sigma(z)$)                                  ▷ once only
            return indices of the elements of $A \cup B$ in $L$
        end if
    end for
    return fail
end function
```

Computing the Polynomial Syndrome: The first polynomial syndrome $R_0(z)$ is computed once only from s. Then, as many times as necessary, $R_0(z)$ is adjusted by computing and adding $\delta = w - t$ elementary syndromes $f_\beta(z)$. This adjustment has a cost proportional to δt field operations which is negligible in practice.

Solving the Key Equation: As mentioned above, there are various syndromes and key equations and sometimes several ways to solve them. In all case this resolution has to be done completely and produces the same locator polynomial $\sigma(z)$. The cost is proportional to t^2 field operations with various constant.

Root Finding: The key equation always has a solution $\sigma(z)$ regardless of the existence of a suitable error pattern of weight t. The error has weight t or less if and only if the polynomial $\sigma(z)$ has all its roots in the field $\mathbf{F}_{2^m}$ that is if $\sigma(z) \mid z^{2^m} - z$. In practice we check whether $z^{2^m} = z \bmod \sigma(z)$ and we only compute the roots once. This requires m polynomial squaring modulo $\sigma(z)$ for a cost proportional to mt^2 field operations. This will be the dominant cost for the signature.

5.3 Discarding Degenerate Instances of Decoding

Several syndromes and key equations may be used for implementing the algebraic decoding of Goppa codes. In all cases, there is some amount of control required; at some point a coefficient is checked (leading coefficient in the extended Euclidean algorithm, or the discrepancy in Berlekamp-Massey Algorithm) and if it is zero the sequencing of operations if affected. Ignoring completely this test (*i.e.* assuming the coefficient is non zero) will provide a significant speedup in software (loops are easier to unroll) and a welcome simplicity in constrained devices. The counterpart is that a (small) proportion of decoding attempts produce inconsistent results and will fail. This is not a big deal in the signature context where almost all decoding attempts fail anyway. This was already remarked in [3] to simplify the control in an FPGA implementation.

5.4 How to Handle Decoding Failure

For $(m, t) = (20, 8)$ and $w = 9$ there is a probability of failure of $\nu = 5.5\%$. This means that some messages cannot be signed. When we use Parallel-CFS with multiplicity $\lambda = 5$, this percentage is equal to $\mu = 1 - (1 - \nu)^\lambda$ that is almost 25%, which is hardly acceptable. The workaround is to *add a counter:*

> Define a family $\mathcal{F} = \{f_i, 0 \leq i < 2^b\}$ of 2^b one-to-one transformations on the syndromes (for instance adding distinct predefined random constant vectors). Let $s_1, \ldots, s_\lambda$ denote the hash values of Parallel-CFS of multiplicity λ. We try to decode the tuple $f(s_1), \ldots, f(s_\lambda)$ for all $f \in \mathcal{F}$. The verifier will also try all $f \in \mathcal{F}$, in the same order as the signer. Optionally, the b bits index i of the transformation can be added to the signature to speed-up the verification.

The decoding fails for all $f \in \mathcal{F}$ with probability μ^{2^b}. For instance with $b = 5$ in our example the probability of failure drops from 0.249 to 2^{-64} and each time we increment b this probability is squared. The security is unchanged because we applied the same transformation on all hash values. Note that, had we allowed different transformations for the λ hash values, the attacker would have been able to apply a 2^b instances ISD-DOOM for on each hash value, gaining a factor $\sqrt{2^b}$ to the attack.

5.5 Signature Size, Verification and Key Generation

Various interesting tradeoffs are possible between signature size and verification time. They are described already in [1] and we propose no novelty here. For $(m, t, w, \lambda) = (20, 8, 10, 3)$ the signature size ranges from 292 to 535 bits and for $(m, t, w, \lambda) = (18, 9, 11, 3)$ it ranges from 286 to 519 bits. This part of the scheme (as well as the key generation procedure) is out of the scope of this work and is not detailed further.

6 Timings

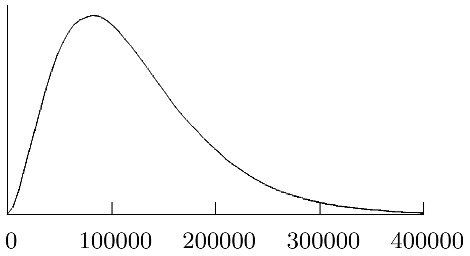
6.1 Computation time

We measured with two decoders and the various sets of parameters the average number of algebraic decoding and the running time for producing one signature running on our target platform (a single core of an Intel Xeon W3670 at 3.20 GHz, about 400 000 signatures per parameter set were computed). The finite field arithmetic primitives use 75% of the total amount of CPU time, most of that (66%) for the sole multiplication. We observe in Table 3 that the number of decoding is very close (but slightly above) the expected value of $\lambda t!$. The only exception is for $(m, t, w, \lambda) = (20, 8, 9, 5)$. In that case each complete decoding has a probability of 5.5% of failure, but when it fails, the number of decoding attempts is equal to the maximum allowed. Experimentally the best tradeoff when $\lambda = 5$ is to allow only 200 000 decoding per binary syndrome (instead of 2^{20}), this raises the probability of failure to 8.4% and with a counter of 6 bits (see §5.4) we fail to sign with probability 2^{-95}. In practice we observe that the signing cost almost doubles.

	(m, t, w, λ)			
	(18,9,11,3)	(18,9,11,4)	(20,8,10,3)	(20,8,9,5)
number of decoding	1 117 008	1 489 344	121 262	360 216
running time (BM)	14.70 s	19.61 s	1.32 s	3.75 s
running time (Pat.)	15.26 s	20.34 s	1.55 s	4.26 s

Table 3. Average number of algebraic decoding and running time per signature

Fig. 1. Distribution of the number of decoding per signature $(m, t, w, \lambda) = (20, 8, 10, 3)$



6.2 Comparing Decoders

We provide here the number of elementary field operations needed for one algebraic decoding attempt. Those numbers were obtained by running the software.

Statistics are summarized in Table 4. The “critical” steps are those called inside the loop of Algorithm 3. The “non critical” ones are outside the loop and thus are called only once per complete decoding, that is $\lambda = 3$ times per signature. A field operation is a multiplication, a squaring, a division, an inversion, or a square root. We do not count additions which are implemented with a xor. In practice, this gives an accurate measure of the complexity and allows and easy comparison of the decoders and an indication about the relative costs of the various steps. All numbers are constant for all steps except the root finding algorithm (Berlekamp trace algorithm). If we consider the non critical parts, it appears that the syndrome computation and the root finding algorithms are the dominant cost and thus the Patterson algorithm is more efficient than the Berlekamp-Massey algorithm which requires a double sized syndrome. The situation is reversed when we consider only the critical parts because the Berlekamp-Massey key equation solving is more efficient.

		critical				non critical	
(m, t)	type	(1)	(2)	(3)	(1)+(2)+(3)	(4)	(5)
(18,9)	BM	58	180	840	1078	2184	3079.1
(18,9)	Pat.	38	329	840	1207	1482	3079.1
(20,8)	BM	52	144	747	943	1950	3024.6
(20,8)	Pat.	34	258	747	1039	1326	3024.6

(1) syndrome adjustment
(2) key equation solving
(3) split checking
(4) initial syndrome
(5) root finding

Table 4. Number of field operations (excluding additions) per decoding

7 Conclusion

For a proper choice of parameters we have shown that CFS, in fact Parallel-CFS, is practical, though cumbersome to achieve a reasonable security. The fastest of our instances needs a bit more than one second of CPU time to produce a signature, which is slow but acceptable. The corresponding public key has a size of 20 megabytes, which may disqualify the scheme for some applications. Note that the public key is not needed for signing but only the secret key which consists of a pair (L, g). The generator g has a size of mt bits (160 or 162 bits here) and the support is a permutation of 2^m elements which can be generated on the fly from a seed. The implementation of the signing primitive we describe requires only a relatively small amount of storage[3] and memory, making it suitable for massively parallel architecture (like GPUs) or “hardware-oriented” devices (like FPGAs or even smart cards).

[3] mostly for the finite field for which there are other options

References

1. Courtois, N., Finiasz, M., Sendrier, N.: How to achieve a McEliece-based digital signature scheme. In Boyd, C., ed.: Advances in Cryptology - ASIACRYPT 2001. Volume 2248 of LNCS., Springer (2001) 157–174
2. McEliece, R.: A public-key cryptosystem based on algebraic coding theory. DSN Prog. Rep., Jet Prop. Lab., California Inst. Technol., Pasadena, CA (January 1978) 114–116
3. Beuchat, J.L., Sendrier, N., Tisserand, A., Villard, G.: FPGA implementation of a recently published signature scheme. Rapport de recherche RR LIP 2004-14, ENS Lyon (2004)
4. Faugère, J.C., Gauthier, V., Otmani, A., Perret, L., Tillich, J.P.: A distinguisher for high rate McEliece cryptosystems. In: ITW 2011, Paraty, Brazil (October 2011) 282–286
5. Finiasz, M.: Parallel-CFS: Strengthening the CFS McEliece-based signature scheme. In Biryukov, A., Gong, G., Stinson, D., eds.: Selected Areas in Cryptography. Volume 6544 of LNCS., Springer (2010) 159–170
6. Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems. IEEE Transactions on Information Theory **24**(3) (May 1978)
7. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions **IT-8** (1962) S5–S9
8. Stern, J.: A method for finding codewords of small weight. In Cohen, G., Wolfmann, J., eds.: Coding theory and applications. Volume 388 of LNCS., Springer (1989) 106–113
9. Bernstein, D., Lange, T., Peters, C.: Smaller decoding exponents: Ball-collision decoding. In Rogaway, P., ed.: Advances in Cryptology - CRYPTO 2011. Volume 6841 of LNCS., Springer (2011) 743–760 To appear.
10. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In Lee, D., Wang, X., eds.: Advances in Cryptology - ASIACRYPT 2011. Volume 7073 of LNCS., Springer (2011)
11. Wagner, D.: A generalized birthday problem. In Yung, M., ed.: Advances in Cryptology - CRYPTO 2002. Volume 2442 of LNCS., Springer (2002) 288–303
12. Camion, P., Patarin, J.: The knapsack hash function proposed at CRYPTO '89 can be broken. In Davies, D., ed.: Advances in Cryptology - EUROCRYPT '91. Volume 547 of LNCS., Springer (1991) 39–53
13. Coron, J.S., Joux, A.: Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2004/013 (2004) http://eprint.iacr.org/.
14. Overbeck, R., Sendrier, N.: Code-based cryptography. In Bernstein, D., Buchmann, J., Dahmen, E., eds.: Post-Quantum Cryptography. Springer (2009) 95–145
15. Johansson, T., Jönsson, F.: On the complexity of some cryptographic problems based on the general decoding problem. IEEE-IT **48**(10) (October 2002) 2669–2678
16. Sendrier, N.: Decoding one out of many. In Yang, B.Y., ed.: PQCrypto 2011. Volume 7071 of LNCS., Springer (2011) 51–67
17. Dumer, I.: On minimum distance decoding of linear codes. In: Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory, Moscow (1991) 50–52
18. Finiasz, M., Sendrier, N.: Security bounds for the design of code-based cryptosystems. In Matsui, M., ed.: Advances in Cryptology - ASIACRYPT 2009. Volume 5912 of LNCS., Springer (2009) 88–105

19. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How 1+1=0 improves information set decoding. In: EUROCRYPT 2012. (2012)
20. Patterson, N.: The algebraic decoding of Goppa codes. IEEE Transactions on Information Theory **21**(2) (March 1975) 203–207
21. Massey, J.: Shift-register synthesis and BCH decoding. IEEE Transactions on Information Theory **15**(1) (January 1969) 122–127
22. Berlekamp, E.: Algebraic Coding Theory. Aegen Park Press (1968)

A A Non Asymptotic Analysis of ISD-MMT

We refer to the algorithm described in [10] to solve $\mathrm{CSD}(H, s, w)$ with $H \in \{0,1\}^{r \times n}$ and $s \in \{0,1\}^r$ (we denote $k = n - r$ the dimension), it uses as parameters three integers p, ℓ_2, and ℓ. First, a Gaussian elimination on H is performed and then three levels of lists are built and successively merged. With a certain probability $\mathcal{P}(p, \ell, \ell_2)$ a solution to $\mathrm{CSD}(H, s, w)$ lies in the last of those lists. The whole process has to be repeated $1/\mathcal{P}(p, \ell, \ell_2)$ times on average to find a solution.

- There are 4 lists at the first level, each of size $L_0 = \binom{(k+\ell)/2}{p/4}$.
- There are 2 lists at the second level, obtained by merging the first level lists pairwise, both have size $L_1 = L_0^2 2^{-\ell_2}$ on average.
- The final list at third level is obtained by merging the two second level lists and has size $L_2 = L_1^2 2^{-\ell+\ell_2} = L_0^4 2^{-\ell-\ell_2}$ on average.

To give an expression of the success probability, we cannot use [10] which assumes a unique solution while for signature parameters we may have several[4]. Instead we claim (following the analysis of [18, 16]) that any particular element of the final list will provide a solution with probability $\approx 2^\ell \varepsilon(p, \ell)$ where

$$\varepsilon(p, \ell) = \frac{\binom{r-\ell}{w-p}}{\min\left(2^r, \binom{n}{w}\right)}.$$

The min in the above expression takes into account the possibility of several solutions. In practice for the signature 2^r will be smaller than $\binom{n}{w}$. We claim that, for practical code parameters and when p and ℓ are near their optimal values

1. if ℓ_2 is not too small the proportion of duplicates in the final list is negligible,
2. if ℓ_2 is not too large the costs for building the first level lists and for the Gaussian eliminations are negligible.

[4] If the weight w is not above the Gilbert-Varshamov radius by more than a constant, the expected number of solution is polynomial and do not affect the asymptotic analysis, for a non-asymptotic analysis the difference is significant

Assuming the first claim is true, the probability of success is

$$\mathcal{P}(p,\ell,\ell_2) = 1 - \left(1 - 2^\ell \varepsilon(p,\ell)\right)^{L_2} \approx L_2 2^\ell \varepsilon(p,\ell) = L_0^4 2^{-\ell_2} \varepsilon(p,\ell).$$

Assuming the second claim is true, the cost for building the final list and checking whether it contains a solution is (crudely) lower bounded by $2L_1+2L_2$ elementary operations[5]. The factor 2 in front of L_1 is because there are two lists at the second level and the factor 2 in front of L_2 is because each element of the final list has to be constructed (an addition at least) then checked (a Hamming weight computation). Finally assuming each elementary operation costs at least ℓ binary operations the cost of ISD-MMT is lower bounded by

$$\mathrm{WF}_{\mathrm{MMT}} = \min_{p,\ell} \frac{2\ell}{\varepsilon(p,\ell)} \left(\frac{1}{2^\ell} + \frac{1}{L_0^2}\right). \tag{6}$$

Note that, interestingly, ℓ_2 do not appear in the above expression. It means that, as long as it is neither too small or too large, the choice of ℓ_2 has no impact on the complexity of ISD-MMT. In practice the proper ranges is (roughly) $p/2 \leq \ell_2 \leq \log_2 L_0$. It is best to avoid the extreme values in that range and large values are better because they reduce memory requirements. Finally note that in [10] it is suggested that $\ell_2 \leq p-2$. This is marginally inside the acceptable range but this has no consequence on the asymptotic exponent analysis.

[5] here an elementary operation is an operation on a column of H, either addition or Hamming weight, possibly with a memory store or read