Toward Secure Implementation of McEliece Decryption

Mariya Georgieva[1] and Frédéric de Portzamparc[1,2,3,4]

Gemalto - Security Lab, Meudon, France[1]
INRIA, Paris-Rocquencourt Center[2],
Sorbonne Universités, UPMC Univ Paris 06, POLSYS,
UMR 7606, LIP6, F-75005, Paris, France[3]
CNRS, UMR 7606, LIP6, F-75005, Paris, France[4]
Mariya.Georgieva@gemalto.com,frederic.urvoy-de-portzamparc@polytechnique.org

Abstract. We analyse the security regarding timing attacks of implementations of the decryption in McEliece PKC with binary Goppa codes. First, we review and extend the existing attacks, both on the messages and on the keys. We show that, until now, no satisfactory countermeasure could erase all the timing leakages in the Extended Euclidean Algorithm (EEA) step. Then, we describe a version of the EEA never used for McEliece so far. It uses a constant number of operations for given public parameters. In particular, the operation flow does not depend on the input of the decryption, and thus closes all previous timing attacks. We end up with what should become a central tool toward a secure implementation of McEliece decryption.

1 Introduction

Context of this work. Code-based cryptography relies on the hardness of *decoding*, that is recovering $\mathbf{m}$ and $\mathbf{e}$ when given only $\mathbf{c} = \mathbf{mG}+\mathbf{e}$ and $\mathbf{G}$ (for $\mathbf{m} \in \mathbb{F}_q^k, \mathbf{G} \in \mathbb{F}_q^{k\times n}$ and $\mathbf{e} \in \mathbb{F}_q^n$). Indeed, decoding has a complexity exponential when k and the error weight grow linearly in n and no structure is known on $\mathbf{G}$ [2]. However, the error weight is critical for security for another reason: contrary to the public parameters of the code which are fixed at set by an external entity, the error may vary at each encryption, and may even be chosen by any public user (in some situations).

Therefore, a problem arises in most of the implementations of McEliece proposed (*e.g.* in [5,6,14,13]) because the operation flow of the decryption is strongly influenced by the error vector, but no information is known about the error vector when starting decryption. From an attacker's point of view, this is a favorable situation. It means that the observed or manipulated device may leak information before any detection of the attack. These security aspects were addressed by various authors, who explained that a device implementing an unprotected decryption is prone to attacks on the messages [12,1] and on the key [15,16].

Although countermeasures were proposed against some of the leakages, the situation is still unsatisfactory, as it is noticed in the conclusion of [16]. In particular, to the best of our knowledge, no decryption algorithm requiring a number of steps independent of the error weight was described. The work of Bernstein et al. in [4] claims to achieve this goal, but some steps of the decryption (including the extended Euclidean algorithm (EEA) in the decoding) are skipped in the description, and no implementation is publicly available.

Our contributions. First, we gather the different weaknesses from [12,15,1,16]. Those attacks targeted only one of the two known methods for decoding a binary Goppa code (namely Patterson algorithm). We evaluate how/if those threats transpose to the other decoding method (*i.e.* the alternant decoder). We detail the attacks of Strenzke and show that they can be extended to bypass the countermeasure of [15]. Our central contribution consists in describing an EEA tailored for the alternant decoder which has a flow of operations independent of the error vectors (Alg. 8). It was inspired by a work of Berlekamp [3]. We explain step-by-step the construction of the algorithm, and provide completeness proofs (which we could not find in the literature) in the full version of this article.

2 McEliece Public-Key Encryption

We recall in Alg. 1 the encryption and decryption in McElice PKC instantiated with a binary Goppa code, that is $q = 2$. The public key is $\mathbf{G}$ a $k \times n$ matrix over $\mathbb{F}_q$ whose rows generate a Goppa code described by the secret elements $\mathbf{x} \in \mathbb{F}_{q^m}^n$ and $g(z) \in \mathbb{F}_{q^m}[z]$ of degree t.

We detail the two possible methods for decoding a binary Goppa code. One uses the fact that Goppa codes belong to the larger class of alternant codes, so we call it the Alternant Decoder. The other one, called Patterson Algorithm, is specific to binary Goppa codes. For both, the inputs are is $\mathbf{c}$ an encoded message $\mathbf{m} \in \mathbb{F}_q^k$ with unkown error $\mathbf{e}$: $\mathbf{c} = \mathbf{mG} + \mathbf{e}$, where the Hamming weight of $\mathbf{e}$

Algorithm 1 McEliece Cryptosystem

PARAMETERS : Field size q, code length n and dimension k, parameters m, t such that $n - mt \leqslant 0$. Plaintext space: $\mathbb{F}_q^k$. Ciphertext space: $\mathbb{F}_q^n$.
KEYGEN : Pick a support $\mathbf{x} \in \mathbb{F}_{q^m}^n$, a polynomial $g \in \mathbb{F}_{q^m}[x]$ of degree t, $\mathbf{G}$ a generator matrix of $\mathscr{G}(\mathbf{x}, g)$.
PUBLIC KEY : $\mathbf{G}_{pub} = \mathbf{SGP}$, t the correction capacity of the code $\mathscr{G}(\mathbf{x}, g)$.
PRIVATE KEY : T_t a t-decoder for $\mathscr{G}(\mathbf{x}, g)$, $\mathbf{S}$ a random full rank $(n-k) \times (n-k)$ matrix , $\mathbf{P}$ a random $n \times n$ permutation matrix.

ENCRYPT :
1: Input $\mathbf{m} \in \mathbb{F}_q^k$.
2: Generate random $\mathbf{e} \in \mathbb{F}_q^n$ with $w_H(\mathbf{e}) = t$.
3: Output $\mathbf{c} = \mathbf{mG}_{pub} + \mathbf{e}$.

DECRYPT :
1: Input $\mathbf{c} \in \mathbb{F}_q^n$.
2: Compute $\bar{\mathbf{m}} = T_t(\mathbf{cP}^{-1}))$.
3: If decoding succeeds, output $\mathbf{S}^{-1}\bar{\mathbf{m}}$, else output $\perp$.

(denoted in the rest of this article by $w_H(\mathbf{e})$) satisfies $w_H(\mathbf{e}) \leqslant t$, and the secret $\mathbf{x}, g(z)$. The output is $\mathbf{e}$. The main steps are :

1. Compute the **polynomial syndrome** $S(z)$, a univariate polynomial deduced from $\mathbf{c}$, but depending only on $\mathbf{e}$.
2. Solve the **key equation**, which is an equation whose unkowns are univariate polynomials, using an EEA. The solutions give access to the **error locator polynomial** $\sigma_{\mathbf{e}}(z)$, whose roots are related to the support elements x_{i_j} in the error positions i_j. It also the yields the **error evaluator polynomial** $\omega_{\mathbf{e}}(z)$ (helpful to find the values of the errors).
3. Find the roots of $\sigma_{\mathbf{e}}(z)$. Here $\mathbf{e} \in \mathbb{F}_2^n$, so $e_{i_j} \neq 0$ implies that $e_{i_j} = 1$.

The polynomial syndromes and key equations are specific to each method.

Polynomial syndrome

$S_{Alt,\mathbf{e}}(z) = \sum_{\ell=0}^{2t-1} \left(\sum_{i=0}^{n-1} c_i g(x_i)^{-2} x_i^\ell \right) z^\ell.$

Polynomials to be recovered

$$\sigma_{inv,\mathbf{e}}(z) = \prod_{j=1}^{w} (1 - z x_{i_j}),$$

$$\omega_{inv,\mathbf{e}}(z) = \sum_{j=1}^{w} e_{i_j} g(x_{i_j})^{-1} \prod_{\substack{s=1 \\ s \neq j}}^{w} (1 - z x_{i_s}).$$

Key equation

$(\sigma_{inv,\mathbf{e}}, \omega_{inv,\mathbf{e}})$ unique solution of

$$\begin{cases} \omega_{inv,\mathbf{e}}(z) = \sigma_{inv}(z) S_{Alt,\mathbf{e}}(z) \mod z^{2t}, \\ \deg(\sigma_{inv}) \leqslant \lfloor t/2 \rfloor, \deg(\omega_{inv}) < \lfloor t/2 \rfloor. \end{cases}$$

Resolution

EEA$(z^{2t}, S_{Alt,\mathbf{e}}, t)$ outputs $(\mu\sigma_{inv}, (-1)^N \mu \omega_{inv}), \mu \in \mathbb{F}_{q^m}^*, N \geqslant 0$.

Error recovery

$\sigma_{\mathbf{e}}(z) = z^w \sigma_{inv}(1/z)$.
Find the roots of $\sigma_{\mathbf{e}}$.

Fig. 1: Alternant Decoder

Polynomial syndrome

$S_{Gop,\mathbf{e}}(z) = \sum_{i=0}^{n-1} \frac{c_i}{z - x_i} \mod g(z)$.

Polynomials to be recovered

$$\sigma_{\mathbf{e}}(z) = \prod_{j=1}^{w} (z - x_{i_j}),$$

$$\omega_{\mathbf{e}}(z) = \sum_{j=1}^{w} \prod_{\substack{s=1 \\ s \neq j}}^{w} (z - x_{i_s}).$$

Key equation

(σ_1, σ_2) unique solution of

$$\begin{cases} \tau(z)\sigma_2(z) = \sigma_1(z) \mod g(z), \\ \deg(\sigma_1) \leqslant \lfloor t/2 \rfloor, \deg(\sigma_2) < \lfloor t/2 \rfloor, \end{cases}$$

$\tau(z) = \sqrt{S_{Gop,\mathbf{e}}(z)^{-1} + z} \mod g(z)$.

Resolution

1.EEA$(g(z), S_{Gop,\mathbf{e}}(z), 0)$
outputs $(S_{Gop,\mathbf{e}}^{-1} \mod g)$,
2.EEA$(g(z), \tau, \lfloor t/2 \rfloor)$
outputs (σ_1, σ_2).

Error recovery

$\sigma_{\mathbf{e}}(z) = \sigma_1(z)^2 + z\sigma_2(z)^2$,
$\omega_{\mathbf{e}} = \sigma_{\mathbf{e}} S_{\mathbf{e}} \mod g$.
Find the roots of $\sigma_{\mathbf{e}}$.

Fig. 2: Patterson Algorithm

Completeness proofs are classic coding theory literature. For details, see for instance [7][Ch. 12 §9] for the Alternant Decoder and [18,9] for Patterson Alg.

Algorithm 2 Extended Euclidean Algorithm (EEA)

Input: $a(z), b(z), \deg(a) \geqslant \deg(b), d_{fin} \geqslant 0$
Output: $u(z), r(z)$ with $b(z)u(z) = r(z) \mod a(z)$ and $\deg(r) \leqslant d$

$r_{-1}(z) \leftarrow a(z), r_0(z) \leftarrow b(z), u_{-1}(z) \leftarrow 1, u_0(z) \leftarrow 0,$
$i \leftarrow 0$
while $\deg(r_i(z)) > d_{fin}$ **do**
  $i \leftarrow i + 1$
  $q_i \leftarrow r_{i-2}(z)/r_{i-1}(z)$, quotient of the Euclidean division of $r_{i-2}(z)$ by $r_{i-1}(z)$
  $r_i \leftarrow r_{i-2}(z) - q_i(z)r_{i-1}(z)$, rest of the Euclidean division of $r_{i-2}(z)$ by $r_{i-1}(z)$
  $u_i \leftarrow u_{i-2}(z) - q_i(z)u_{i-1}(z)$
end while
$N \leftarrow i$
return $u_N(z), r_N(z)$

The EEA which is used in all the available implementations (see [5,6,14,13]) consists in successive Euclidean divisions as in Alg. 2. Its complexity is $O(\deg(a)^2)$ field multiplications. Asymptotically better algorithms exist, generally referred to as Fast EEA or HGCD (for Half-GCD), with complexity $O(\deg(a) \log \deg(a))$. The reason not to use them here is that constants are hidden in the O (see for details [19]), so that for practical McEliece parameters ($t \leqslant 200$), those are not more efficient than Alg. 2.

The EEA executions solving the key equations have complexities of $7.5tw_H(\mathbf{e})$ (Alternant decoder) and $3.5tw_H(\mathbf{e})$ (Patterson alg.), as shown in [18][§5]. The complexity of the syndrome polynomial inversion (first EEA in Patterson Alg.) can be bounded by $2t^2$. We obtain, for a weight t error, a cost in field multiplications of $7.5t^2$ for the Alternant decoder and $5.5t^2$ for Patterson algorithm. This is why Patterson algorithm is generally preferred.

3 Decryption oracle attacks

3.1 Plaintext-recovery attacks

The attacker has a ciphertext $\mathbf{c}$ and has decryption oracle: he can request and observe the decryption of any message $\mathbf{c}' \neq \mathbf{c}$. In [17,12,1,15], the authors described attacks using the same idea (Alg. 3).

They exploit a decryption oracle to recover the plaintext from an encrypted message $\mathbf{c}$. They focus on Patterson algorithm (Fig. 2) and propose [12][Alg. 5, p. 171], a modified EEA which takes same execution time both on the ciphertext $\mathbf{c}$ and on the twisted $\mathbf{c}^{\star i}$.

EEA leakages in the Alternant Decoder. We adapt the framework of Alg. 3 to the alternant decoder. The alternant decoder, as Patterson one, resorts to an EEA (Fig. 1) prone to leak information, since it execution time depends on the degree of the output, so on the error weight. Table 1 gives the link of the weight of the error vector with the degree of the output of the EEA in Alg. 1.

Algorithm 3 Framework for message-recovery attacks on a decryption device.

INPUT: A valid ciphertext $\mathbf{c} = \mathbf{m}\mathbf{G}_{pub} + \mathbf{e}$, a decryption device $\mathcal{D}$.
OUTPUT: The error vector $\mathbf{e}$.

1: **for** $i = 0, \ldots, n-1$ **do**
2: Modify $\mathbf{c}$ into $\mathbf{c}^{\star i} = \mathbf{c} + (0, \ldots, 0, \underbrace{1}_{i-\text{th bit}}, 0, \ldots)$ and request decryption $\mathcal{D}(\mathbf{c}^{\star i})$.
3: Deduce by timing analysis or power consumption of $\mathcal{D}$ whether $e_i = 0$ or $e_i = 1$.
4: **end for**
5: **return** Error $\mathbf{e}$.

	$\deg(\sigma_{inv})$ if $e_\alpha = 0$	$\deg(\sigma_{inv})$ if $e_\alpha = 1$
$w_H(\mathbf{e}) = t$	t	$t-1$
$w_H(\mathbf{e}^\star) = t + 1 (e_i = 0)$	t	$t-1$
$w_H(\mathbf{e}^\star) = t - 1 (e_i = 1)$	$t-1$	$t-2$

Table 1: Degrees of the output σ_{inv} of EEA$(z^t, S_{\mathbf{e}}(z), \lfloor t/2 \rfloor)$.($\alpha$ denotes the position of the support such that $x_\alpha = 0$)

After computing a polynomial $\sigma_{inv}(z)$ of degree d, if 0 belongs to the support, there are two possibilities, either the index α such that $x_\alpha = 0$ is not an error position, $\sigma_{\mathbf{e}}$ is not divisible by z, then $\deg(\sigma_{\mathbf{e}}) = \deg(\sigma_{inv})$ and $\sigma_{\mathbf{e}}(z)$ is equal to $z^{\deg(\sigma_{inv})}\sigma_{inv}(z^{-1})$, or α is an error position, and $\sigma_{\mathbf{e}}(z) = z^{\deg(\sigma_{inv})+1}\sigma_{inv}(z^{-1})$. So in this case, looking at $\deg(\sigma_{inv})$ does not distinguish manipulated ciphertexts from correct ones, and the EEA cannot be correctly protected by this method.

Countermeasure. Building up on the countermeasure for Patterson decoding described in [12], we propose the following adaptation (Alg. 4) to the alternant decoder. It always detects ciphertext manipulation provided that 0 is not an element of the support, and somehow restores a usual behavior of the EEA (that is, that of a valid ciphertext). The final output will not be the correct plaintext, but this is not a problem as long as the attacker cannot extract information from this result. However, we note that this protection has the same drawbacks as its Patterson equivalent: each **while** execution does not have same execution time.

Algorithm 4 Protected EEA for Alternant decoder (completes Alg. 2)

7: $v_i \leftarrow v_{i-2}(z) - q_i(z)v_{i-1}(z)$
8: **if** $\deg(r_i) < t$ **then**
9: Manipulate r_i so that $\deg(r_i) = \deg(r_{i-1}) - 1$ (*e.g.* $r_i \leftarrow r_i + z^{\deg(r_{i-1})-1}$).
10: **end if**

3.2 Secret decryption key recovery attacks

We address physical attacks initiated by Strenzke in [15,16] against McEliece encryption using Patterson decoding. It aims at recovering the secret key.

Generic attack scenario. The attack scenario is the following. The attacker has acces to a decryption device $\mathcal{D}$ on which he can perform physical measurements. He also knows a public encryption key, so that he can generate codewords with errors of his choice. By observing the decryption phase (more precisely, the EEA execution), Strenzke shows that one can deduce information on the support elements corresponding to the error positions. Roughly, the reason is that when a polynomial condition on those elements is satisfied, the number of iterations of the **while** loop in Alg. 2 is reduced compared to the average number of iterations necessary to perform the EEA for error vectors of same weight. The attack consists in scanning a lot of error positions and collect sufficiently many polynomial relations so that the algebraic system obtained can be solved.

Alg. 5 sums up the global attack framework arising from [16]. In practice, the polynomials P_w will be, for an error $\mathbf{e} = (0, \ldots, e_{i_1}, \ldots, e_{i_w}, \ldots, 0)$ with $w_H(\mathbf{e}) = w$ and $j \geqslant 0$, the evaluation of the j^{th} elementary symmetric polynomial in w variables in $(x_{i_1}, \ldots, x_{i_w})$, that is:

$$\omega_j(\mathbf{e}) = \sum_{1 \leqslant \ell_1 < \cdots < \ell_j \leqslant w} x_{i_{\ell_1}} \ldots x_{i_{\ell_j}}.$$

Algorithm 5 Framework for key-recovery attacks on a decryption device.

INPUT: A decryption device $\mathcal{D}$, public encryption key $\mathbf{G}_{pub}$.
OUTPUT: The secret support $\mathbf{x}$.

for w well-chosen error weights **do**
  for $(i_1, \ldots, i_w)$ subset of $\{0, \ldots, n-1\}$ **do**
    Pick an error vector $\mathbf{e} = (0, \ldots, e_{i_1}, \ldots, e_{i_w}, \ldots, 0)$ with $w_H(\mathbf{e}) = w$.
    Request decryption $\mathcal{D}(\mathbf{e})$ and perform timing or power consumption analysis.
    if EEA execution faster than average (precise conditions in this Section) **then**
      Deduce a polynomial condition on $x_{i_1}, \ldots, x_{i_w}$ (P_w is a polynomial depending only on w):

$$P_w(x_{i_1}, \ldots, x_{i_w}) = 0 \quad (1)$$

    end if
  end for
end for
Solve the non-linear system of all the collected equations (1).
return Secret support $\mathbf{x} = (x_0, \ldots, x_{n-1})$.

State-of-the-art. More precisely, Strenzke uses errors of weights $w = 1, w = 4$ and $w = 6$. For $w = 6$, errors such that Eq. (1) is satisfied are harder to find

than for $w = 4$. For this reason, his strategy consists in collecting as many Eq. (1) with $w = 1$ and $w = 4$ as possible. He obtains a linear system of rank $n - m$ in the n elements of the support. Then, he selects subsets of errors of weight $w = 6$ to look for Eq. (1). These subsets are chosen so as help the polynomial system solving. According to Strenzke, for an encryption scheme with parameters $m = 10, n = 2^m, t = 40$, it takes about 15,000,000 decryption queries to collect enough equations and 28 hours to solve the algebraic system. Eventually, the full secret support $\mathbf{x}$ is recovered by the attacker, and then the Goppa polynomial is easy to find. Indeed, it is well explained in [8][p. 125] how, given the public key, it is possible to recover one from the other in polynomial time.

First example of leakage exploitable by Framework 5. The first attack resorting to the method of Alg. 5 was proposed by Strenzke in [15]. It focuses on the second EEA of Patterson alg. with errors of weight $w = 4$. In this case, $S_{\mathbf{e}}(z) = \sum_{j=1}^{4} \frac{1}{z - x_{i_j}} = \frac{\omega_{\mathbf{e}}(z)}{\sigma_{\mathbf{e}}(z)}$, and

$$\omega_{\mathbf{e}}(z) = \underbrace{(x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4})}_{\omega_1(\mathbf{e})} z^2 + \underbrace{x_{i_1}x_{i_2}x_{i_3} + x_{i_1}x_{i_2}x_{i_4} + x_{i_1}x_{i_3}x_{i_4} + x_{i_2}x_{i_3}x_{i_4}}_{\omega_3(\mathbf{e})}.$$

If $\omega_1(\mathbf{e}) = 0$, then $S_{\mathbf{e}}(z) = \frac{\omega_3(\mathbf{e})}{\sigma_{\mathbf{e}}(z)}$, and $S_{\mathbf{e}}^{-1} \mod g = \omega_3(\mathbf{e})^{-1}\sigma_{\mathbf{e}}(z)$ therefore $\tau(z) = \sqrt{S_{\mathbf{e}}^{-1}(z) + z} \mod g(z) = \sqrt{\omega_3(\mathbf{e})^{-1}\sigma_{\mathbf{e}}(z) + z}$ and $\tau(z)$ has degree lower than $\lfloor t/2 \rfloor$ (for $w = 4$ we have $\deg(\tau(z)) = 2$). As a consequence, the **while** test in $\mathrm{EEA}(g(z), \tau(z), \lfloor t/2 \rfloor)$ is never fulfilled and the number of iterations N is equal to 0. When $\omega_1(\mathbf{e}) \neq 0$, $\deg(\tau(z)) > \lfloor t/2 \rfloor$ with overwhelming probability ($\tau(z)$ is a reduction modulo a polynomial of degree t), so that $N > 0$. This allows to collect many equations of the form $x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4} = 0$. As Strenzke explains, the final system's rank never exceeds $n - m$. So it is not sufficient in practice to fully recover the private key. Still, he proposes a counter-measure.

Counter-measure to protect Second EEA by Strenzke. [15][Sect. 5], Strenzke proposes to detect the polynomials $\tau(z)$ leading to this leakage by checking if $\deg(\tau(z)) < \lfloor t/2 \rfloor$. This can be done just after the determination of $\tau(z)$. If so, manipulate $\tau(z)$ so that is has degree $t - 1$. This countermeasure avoids leaking information only in the second EEA, only when decoding errors of weight 4. Exploitable leakages remain, as shown in the next paragraph.

Leakage in the first EEA of Patterson Decoding. In order to complete the attack initiated in [15], Strenzke proposed in [16] to apply Alg. 5 by focusing on time leakages in both EEA's of Patterson decoding. In [16][Corollary 1], he gives the number of iterations of the **while** loop in the first EEA. We recall it here, and complete it with the analogous result for the second EEA.

Lemma 1. *Let $\mathscr{C} = \mathscr{G}(\mathbf{x}, g(z))$ be a binary Goppa code and $S_{\mathbf{e}}(z)$ the polynomial syndrome associated to an error $\mathbf{e}$ with $w_H(\mathbf{e}) \leqslant \deg(g)/2 - 1$. Write $S_{\mathbf{e}}(z) =$*

$\frac{\omega_{\mathbf{e}}(z)}{\sigma_{\mathbf{e}}(z)} \mod g(z)$. *Let* N_I *and* N_K *be the number of iterations of the* **while** *loop respectively in* $\mathrm{EEA}(g(z), S_{\mathbf{e}}(z), 0)$ *and* $\mathrm{EEA}(g(z), \tau(z), \lfloor t/2 \rfloor)$. *Then*

$$N_I \leqslant \deg(\omega_{\mathbf{e}}(z)) + \deg(\sigma_{\mathbf{e}}(z)) \text{ and } N_K \leqslant \deg(\omega_{\mathbf{e}}(z))/2. \tag{2}$$

Proof. The result on N_I is proved in [16][Corollary 1]. Regarding N_K, observe that v_0 has degree 0 and $v_{N_K} = \sigma_2(z)$ has degree $\deg(\omega_{\mathbf{e}})/2$ (since by derivating the relation $\sigma = \sigma_1^2 + z\sigma_2^2$ we obtain $\omega_{\mathbf{e}} = \sigma_2^2$). As the degrees are raised at least by one at each iteration, we obtain $N_K \leqslant \deg(\omega_{\mathbf{e}})/2$.

Errors weights $w = 4$. Pick $\mathbf{e} = (0, \ldots, e_{i_1}, \ldots, e_{i_4}, \ldots, 0)$. We have $\omega_{\mathbf{e}}(z) = \omega_1(\mathbf{e})z^2 + \omega_3(\mathbf{e})$. According to Lemma 1, N_I satisfies

$$\begin{aligned} x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4} \neq 0 &\implies N_I = 6, \\ x_{i_1} + x_{i_2} + x_{i_3} + x_{i_4} = 0 &\implies N_I = 4. \end{aligned}$$

Therefore, even if the second EEA has been protected with Strenzke's counter-measure, errors of weight $w = 4$ leak the same information in the first EEA. Other equations are found by using error with weight $w = 6$.

Error weights $w = 6$. For $\mathbf{e} = (0, \ldots, e_{i_1}, \ldots, e_{i_6}, \ldots, 0)$, we have for $S_{Gop,\mathbf{e}}(z)$:

$$S_{Gop,\mathbf{e}}(z) = \frac{\omega_1(\mathbf{e})z^4 + \omega_3(\mathbf{e})z^2 + \omega_5(\mathbf{e})}{\sigma_{\mathbf{e}}(z)}.$$

Strenzke's purpose is to detect for which $\mathbf{e}$ is holds that $\omega_3(\mathbf{e}) = \omega_1(\mathbf{e}) = 0$. These cases are exactly those with $S_{\mathbf{e}}(z)^{-1} = \omega_5(\mathbf{e})^{-1}\sigma_{\mathbf{e}}(z)$ and hence $\deg(\tau(z)) < \lfloor t/2 \rfloor$, so that $N_K = 0$ provided that Strenzke's counter-measure is not applied. This is a somehow surprising proposition, since this criterion can be rendered useless by a counter-measure already proposed by the same author.

		$\mathrm{EEA}(g, S_{\mathbf{e}}, 0)$	$\mathrm{EEA}(g, \tau, \lfloor t/2 \rfloor)$
$w_H(\mathbf{e}) = 4$	$\omega_1(\mathbf{e}) \neq 0$	$N_I \leqslant 6$	$N_K \leqslant 1$
	$\omega_1(\mathbf{e}) = 0$	$N_I \leqslant 4^*$	$N_K = 0^+$ CM $\deg(\tau) < \lfloor t/2 \rfloor^+$
$w_H(\mathbf{e}) = 6$	$\omega_1(\mathbf{e}) \neq 0$, $\omega_3(\mathbf{e}) \neq 0$	$N_I \leqslant 10$	$N_K \leqslant 2$
	$\omega_1(\mathbf{e}) = 0$, $\omega_3(\mathbf{e}) \neq 0$	$N_I \leqslant 8$	$N_K \leqslant 1$
	$\omega_1(\mathbf{e}) = 0$, $\omega_3(\mathbf{e}) = 0$	$N_I \leqslant 6$	$N_K = 0^*$ CM $\deg(\tau) < \lfloor t/2 \rfloor$
$w_H(\mathbf{e}) = 2w'$	$\omega_1(\mathbf{e}) \neq 0$, $\omega_3(\mathbf{e}) \neq 0$	$N_I \leqslant 4w' - 2$	$N_K \leqslant w' - 1$
	$\omega_1(\mathbf{e}) = 0$, $\omega_3(\mathbf{e}) \neq 0$	$N_I \leqslant 4w' - 4$	$N_K \leqslant w' - 2$
	$\omega_1(\mathbf{e}) = 0$, $\omega_3(\mathbf{e}) = 0$	$N_I \leqslant 4w' - 6$	$N_K \leqslant w' - 3$

Table 2: Overview of small- error-weight message attacks. Cases marked with a * or a $^+$ are proposed resp. in [15] and [16].

Combination of first and second EEA. When using error weights $w \geqslant 6$, the attacker will encounter problems due to the fact that all the values given in Table 2 (on page 8) are only bounds (except in the cases $N \leqslant 0$). Indeed, it may happen that one of the Euclidean divisions entails a degree fall greater than 1 independantly of the degree of $\omega_{\mathbf{e}}$. For example, with $w = 6$, the attacker may observe $N_K = 1$ whereas $\omega_1(\mathbf{e})$ is not zero. This remark leads Strenzke to discard those cases for an attack as long as no way of distinguishing thoses cases is found. We propose such distinguisher, by using N_I to determine if $\omega_1(\mathbf{e})$ is zero, as $\omega_1(\mathbf{e}) = 0$ implies $N_I \leqslant 8$. Indeed, an attacker observing the errors $\mathbf{e}$ with $(N_I, N_K) = (10, 1)$ can conclude that $\omega_1(\mathbf{e}) \neq 0$ (Table 2). We may have $(N_I, N_K) = (8, 1)$ when $\omega_1(\mathbf{e}) \neq 0$ if three cancellations occur in the 12 intermediate polynomials, which has probability $p_3 = \binom{12}{3} 2^{-3m}(1 - 2^{-m})^9 \approx 2.10^{-7}$ for $m = 10$ (we model the leading coefficients as random elements of $\mathbb{F}_{2^m}$). When sampling x error vectors, we expect to find $p_3 x$ such misleading cases. With the numbers of samples from [16][Table 2], the probability to find one is not negligible. If at least one wrong equation is deduced, the system to solve has no solution and the attack fails. We propose to avoid this problem by using errors with $w \geqslant 8$.

Error weights $w = 8$. We sampled randomly 10,000,000 errors $\mathbf{e}$ of weight 8 and collected the couples (N_I, N_K) in Table 3. When $w_H(\mathbf{e}) = 8$, there are more possibilities than with $w = 6$. Samples with $(N_I \leqslant 12, N_K \leqslant 2)$ do not necessarily have $\omega_1(\mathbf{e}) = 0$: this happens with probability $p'_3 = \binom{17}{3} 2^{-3m}(1 - 2^{-m})^{14} \approx 6.10^{-7}$ for $m = 10$ (we found 3). In particular, the case marked with a * in Table 3 would make the attacker to think erroneously that the corresponding error vector satisfies $\omega_1(\mathbf{e}) = 0$. However, the number of parasitic cancellations necessary to provide values (N_I, N_K) compatible with $(\omega_1(\mathbf{e}), \omega_3(\mathbf{e})) = (0, 0)$ is 6, which happens with probability $p'_3 = \binom{17}{6} 2^{-6m}(1 - 2^{-m})^{11} \approx 10^{-14}$ for $m = 10$. If $\omega_1(\mathbf{e}) = 0$ but $\omega_3(\mathbf{e}) \neq 0$, then a couple $(10, 1)$ is found if 3 cancellations occur. This has probability $2^{-m} p'_3 \approx 6.10^{-10}$ (as ω_1 takes all the values of $\mathbb{F}_{2^m}$ with same probability). Therefore, we are able to say without ambiguity when $(\omega_1(\mathbf{e}), \omega_3(\mathbf{e})) = (0, 0)$ on a considerable amount of samples. We deduce from our

	No parasitic cancellation	1 parasitic cancellation	2 parasitic cancellations	3 parasitic cancellations
$\omega_1(\mathbf{e}) \neq 0$ $\omega_3(\mathbf{e}) \neq 0$	(14,3): 9855087	(13,3): 115439 (14,2): 18916	(12,3): 614 (13,2): 248 (14,1): 8	(12,2): 1 * (11,3): 2
$\omega_1(\mathbf{e}) = 0$ $\omega_3(\mathbf{e}) \neq 0$	(12,2): 9570	(11,2): 96 (12,1): 8	(10,2): 0 (11,1): 0	
$\omega_1(\mathbf{e}) = 0$ $\omega_3(\mathbf{e}) = 0$	(10,1): 10	(9,1): 0	(8,1): 0	

Table 3: Number of samples for each (N_I, N_K) for 10,000,000 error vectors with $w = 8$. Code parameters: $m = 10, n = 2^m, t = 40$. See text for explanation on *.

samples 10 equations $\omega_1(\mathbf{e}) = 0$ which are correct with proba. $(1 - 10^{-7})$ and 10 equations $\omega_3(\mathbf{e}) = 0$ correct with proba. $(1 - 10^{-3})$. To conclude, although our method requires more samples than the previous one (around 10^9 to collect some thousands equations with ω_1, and dozens with ω_3), we can recover information on the support even if the countermeasure $\deg(\tau) < \lfloor t/2 \rfloor$ is implemented.

Small weight error messages in Alternant decoder We determine if an attacker can retrieve any information by applying Alg. 5 if the Alternant decoder is implemented. First, we give in Lemma 2 the analogous of Lemma 1.

Lemma 2. *Let* $\mathbf{e}$ *be an error with* $w_H(\mathbf{e}) \leqslant t$*. Then* $S_{Alt,\mathbf{e}}(z) = \frac{\omega_{inv,\mathbf{e}}(z)}{\sigma_{inv,\mathbf{e}}(z)} \mod z^{2t}$ *and the number of iterations* N *of the* **while** *loop of the Alternant decoder in the EEA satisfies*

$$N \leqslant N_{max} = \min(\deg(\sigma_{inv,\mathbf{e}}), \deg(S_{Alt,\mathbf{e}}) - \deg(\omega_{inv})). \tag{3}$$

Specific case of weight 1 errors. If $w = 1$, we always have $\deg(\omega_{inv}) = 0$ and $\deg(\sigma_{inv}) = 1$ except if $x_{i_1} = 0$. Indeed, in this case, the polynomial syndrome is a constant: $S_{\mathbf{e}}(z) = \frac{1}{g(0)^2}$ and the **while** loop is never executed.

Error weights $w > 1$. We suppose that no error occurred in the zero element of the support so that $\deg(\sigma_{inv}) = w_H(\mathbf{e})$ always holds (the coefficient of z^w in σ_{inv} is $x_{i_1} \dots x_{i_w}$). Therefore, faster decryptions indicate the cancellation of a leading coefficient in the intermediate values, but in the alternant decoder we found no way of determining which intermediate value was concerned. If by any chance a power analysis can ensure that it is the first intermediate polynomial (that is, the syndrome polynomial $S_{Alt,\mathbf{e}}(z)$) that has a degree smaller than expected, then the information recovered would be:

$$\sum_{j=1}^{w} g(x_{i_j})^{-2} \sum_{j=1}^{w} x_{i_j}^{2t-1} = 0. \tag{4}$$

We observe that the equations written thanks to this method are more complex than with Patterson algorithm, at least for two reasons. First, they are not directly polynomial, and the degrees implied are much higher. Second, as both $\mathbf{x}$ and g have to be unknown ([8][p. 125]), additive unknowns are necessary: either $t + 1$ to describe the secret polynomial's coefficients, or n if we introduce new equations $y_i = g(x_i)^{-2}$. We conclude that the alternant decoder is intrinsically more resistant to Strenzke's attacks. However, the overall security is still not clear due to the uncertainty on the countermeasure (Alg. 4) against Alg. 3.

4 Extended euclidean Algorithm with constant flow

We expose a way of implementing the EEA algorithm unused so far for McEliece decryption. It has the very interesting property of requiring a number of operations depending only on the Goppa polynomial degree t and **not** on the weight

of the error introduced in the ciphertext. Therefore, the attacks of 3.1 and 3.2 are not possible.

It is inspired by Berlekamp's work in [3] (which as followed by other works of optimization in the VLSI community, amongst many others [10,11]). We could find no reference to it in any paper related to McEliece. On the contrary, designing such an algorithm is desirable goal according to the conclusion of [16]. The reason may be that [3] has a very limited access, and we could find no completeness proofs of the algorithm proposed. Here, we transform smoothly the original EEA (Alg. 2) into successive version gaining in regularity (Algorithms 6 and 7). We end up with Alg. 8, which is simpler and more regular than all the previous ones. At each step, we give and prove (in the full version of this article) the form of the outputs and intermediate values. Finally, each execution of Alg. 8 costs, in field multiplications, exactly $16t^2$ ($2t$ times a loop costing $4 \times 2t$).

In the rest of this article we will set N be the number of Euclidean divisions performed during $\mathrm{EEA}(z^{2t}, S_{Alt}(z), t)$ in Alg. 2, $d_i = \deg(r_i(z))$, and $\delta_i = \deg(q_i(z)) = \deg(r_{i-2}) - \deg(r_{i-1})$. For any polynomial $P(z) \in \mathbb{F}_{q^m}[z]$, we denote its coefficients by P_j even for $j > \deg(P)$ (in which case $P_j = 0$), so that

$$P(z) = \sum_{j=0}^{+\infty} P_j z^j = P_{\deg(P)} z^{\deg(P)} + \cdots + P_0.$$

Regarding the δ_i's, we have:

$$\sum_{i=1}^{N} \delta_i = \deg(u_N(z)) = \deg(\omega_{inv,\mathbf{e}}) = w_H(\mathbf{e}) - 1.$$

Unrolling Euclidean divisions In Alg. 6, we decompose each Euclidean division into a number of polynomial subtractions depending only on δ_i the degrees of the quotients. We explicit the intermediate values of the Euclidean division of $R_{i-2}(z)$ by $R_{i-1}(z)$, that we denote by $R_i^{(0)}(z), \ldots, R_i^{(\delta_i+1)}(z)$. To do so, we eliminate in each $R_i^{(j)}(z)$ (for $0 \leqslant j \leqslant \delta_i + 1$) the term $z^{d_{i-2}-j}$, whether the associated coefficient is zero or not. This is why we perform the Euclidean divisions in a way to avoid the divisions by a field elements (Steps 7 to 11 of Alg. 6). Consequently, the outputs are multiple of the outputs of Alg. 2.

Proposition 1 (Comparison of Algorithms 2 and 6). *Let $a(z)$ and $b(z)$ be two polynomials with $\deg(a(z)) \geqslant \deg(b(z))$, and d a non-negative integer. $u_i(z), v_i(z), r_i(z), q_i(z)$ are the intermediate values in Alg. 2, and $U_i(z), V_i(z), R_i(z)$ are the intermediate values in Algorithm 6. It holds that, for all $i = -1, \ldots, N$, there exists $\lambda_i \in \mathbb{F}_{q^m}^*$ such that:*

$$R_i(z) = \lambda_i r_i(z),$$
$$U_i(z) = \lambda_i u_i(z).$$

Hence, $\Delta_i = \deg(R_{i-2}) - \deg(R_{i-1}) = \deg(r_{i-2}) - \deg(r_{i-1}) = \delta_i$ for all i.

Algorithm 6 EEA with unrolled Euclidean Division

Input: $a(z) = z^{2t}, b(z) = S_{\mathbf{e}}(z)$.
Output: $U(z) = \lambda_N \sigma_{\mathbf{e}}(z), R(z) = \lambda_N \omega_{\mathbf{e}}(z)$ (for some $\lambda_N \in \mathbb{F}_{q^m}^*$).

1: $R_{-1}(z) \leftarrow a(z), R_0(z) \leftarrow b(z)$,$U_{-1}(z) \leftarrow 1, U_0(z) \leftarrow 0$, $i \leftarrow 0$.
2: **while** $\deg(R_i(z)) > t$ **do**
3: $i \leftarrow i + 1$
4: $R_{i-2}^{(0)}(z) \leftarrow R_{i-2}(z)$,$U_{i-2}^{(0)}(z) \leftarrow U_{i-2}(z)$
5: $\Delta_i \leftarrow \deg(R_{i-2}) - \deg(R_{i-1})$
6: $\beta_i \leftarrow \mathrm{LC}(R_{i-1}(z))$.
7: **for** $j = 0, \ldots, \Delta_i$ **do**
8: $\alpha_{i,j} \leftarrow R_{i,d_{i-2}-j}^{(j)}$,
9: $R_{i-2}^{(j+1)}(z) \leftarrow \beta_i R_{i-2}^{(j)}(z) - \alpha_{i,j} z^{\Delta_i - j} R_{i-1}(z)$
10: $U_{i-2}^{(j+1)}(z) \leftarrow \beta_i U_{i-2}^{(j)}(z) - \alpha_{i,j} z^{\Delta_i - j} U_{i-1}(z)$
11: **end for**
12: $R_i(z) \leftarrow R_{i-2}^{(\Delta_i+1)}(z)$,$U_i(z) \leftarrow U_{i-2}^{(\Delta_i+1)}(z)$
13: **end while**
14: $N \leftarrow i$.
15: **return** $U_N(z), R_N(z)$

There are two problems with Alg. 6. The first one is that the inner **for** loop (Steps 7 to 11) has a variable length, and contains a multiplication $z^{\delta_i-(j-1)}R_i(z)$ which depends on the iteration, which will produce a recognizable pattern. The second problem is that the **while** loop leads to a number of operations depending on the input. Alg. 7 is a first step towards the resolution of the second problem. It is not realistic (it requires to know the δ_i's), but it eases the proofs of completeness of Alg. 8, which solves both issues.

Regular polynomial shift pattern. In Alg. 7, we perform the Euclidean division in such a way that we only multiply the operand by z at each **for** iteration. This can be done by splitting in two phases each Euclidean divisions. The first phase (Steps 4 to 7) "re-aligns" the operands $\tilde{R}_{i-2}$ and $\tilde{R}_{i-1}$ so that they both have same degree $d = \deg(R_{-1}(z))(= 2t)$. Doing so, the second phase (Steps 8 to 12) compute the polynomial subtractions (corresponding to Steps 9-10 of Alg. 6) and perform a shift "re-aligning" the operands. A consequence is that the polynomials $\tilde{R}_i(z)$ are of the form $z^{k_i}R_i(z)$ and the degrees d_i are lost.

Proposition 2 (Comparison of Algorithms 6 and 7). *For each $i = 1, \ldots, N$, after Step 13 of Alg. 7, it holds that*

$$(\tilde{R}_{i-1}(z), \tilde{R}_i(z)) = (z^{d-d_{i-1}}R_{i-1}(z), z^{d-d_{i-1}+1}R_i(z)),$$
$$(\tilde{U}_{i-1}(z), \tilde{U}_i(z)) = (z^{d-d_{i-1}}U_{i-1}(z), z^{d-d_{i-1}+1}U_i(z)).$$

Complete Regular Flow EEA. To design a real constant flow algorithm, we merge the loops L_1 and L_2 in a common pattern so as to be indistinguishable (Steps 5 to 7 of Alg.8). They differenciate by the assignements which are

Algorithm 7 Toy EEA with regular shift pattern

Input: $a(z) = z^{2t}, b(z) = S_{\mathbf{e}}(z), d = 2t$
Output: $\tilde{U}_N(z) = z^{d-d_{N-1}+1}\sigma_{\mathbf{e}}(z), \tilde{R}_N(z) = z^{d-d_{N-1}+1}\omega_{\mathbf{e}}(z)$.

1: $\tilde{R}_{-1}(z) \leftarrow a(z), \tilde{R}_0(z) \leftarrow zb(z), \tilde{U}_{-1}(z) \leftarrow 1, \tilde{U}_0(z) \leftarrow 0$.
2: **for** $i = 1, \ldots, N$ **do**
3: $\quad \tilde{R}^{(0)}_{i-2}(z) \leftarrow \tilde{R}_{i-2}(z), \tilde{U}^{(0)}_{i-2}(z) \leftarrow \tilde{U}_{i-2}(z)$
4: $\quad$ **for** $j = 1, \ldots, \Delta_i - 1$ **do** $\Big\}$ L_1
5: $\quad\quad \tilde{R}_{i-1}(z) \leftarrow z\tilde{R}_{i-1}(z)$
6: $\quad\quad \tilde{U}_{i-1}(z) \leftarrow z\tilde{U}_{i-1}(z)$
7: $\quad$ **end for**
8: $\quad$ **for** $j = 0, \ldots, \Delta_i$ **do** $\Big\}$ L_2
9: $\quad\quad \tilde{\alpha}_{i,j} \leftarrow \tilde{R}^{(j)}_{i,d}$, $\tilde{\beta}_i \leftarrow \tilde{R}_{i-1,d}$.
10: $\quad\quad \tilde{R}^{(j+1)}_{i-2}(z) \leftarrow z\left(\tilde{\beta}_i\tilde{R}^{(j)}_{i-2}(z) - \tilde{\alpha}_{i,j}\tilde{R}_{i-1}(z)\right)$
11: $\quad\quad \tilde{U}^{(j+1)}_{i-2}(z) \leftarrow z\left(\tilde{\beta}_i\tilde{U}^{(j)}_{i-2}(z) - \tilde{\alpha}_{i,j}\tilde{U}_{i-1}(z)\right)$
12: $\quad$ **end for**
13: $\quad \tilde{R}_i(z) \leftarrow \tilde{R}^{(\Delta_i+1)}_{i-2}(z), \tilde{U}_i(z) \leftarrow \tilde{U}^{(\Delta_i+1)}_{i-2}(z)$
14: **end for**
15: **return** $\tilde{U}_N(z), \tilde{R}_N(z)$

performed in Steps 14- 15 and 18-19. To know when polynomials substractions have to be stopped, we collect in a counter δ the number of shifts necessary to re-align the operands. Finally, when the polynomials σ_{inv} and ω_{inv} have been computed, the extra executions of the main loop (Steps 4 to 22) consist in shifting the operands. therefore, the number of iterations can be safely set to the maximum value (*ie* $2t$ to decode the errors with $w_H(\mathbf{e}) = t$)), and the **while** loop is replaced by **for**.

Proposition 3 (Comparison of Algorithms 6 and 8.). *For each $i = 1, \ldots, N$, after steps 21, it holds that:*

$$\hat{R}_{2(\delta_1+\cdots+\delta_i)}(z) = z^{d-d_{i-1}+1}R_i(z),$$
$$\hat{U}_{2(\delta_1+\cdots+\delta_i)}(z) = z^{d-d_{i-1}+1}U_i(z).$$

The outputs of Alg. 8 are, for some $\mu \in \mathbb{F}^*_{q^m}$:

$$\hat{R}_d(z) = z^{d-w_H(\mathbf{e})+1}R_N(z) = \mu z^{d-w_H(\mathbf{e})+1}\omega_{inv}(z),$$
$$\hat{U}_d(z) = z^{d-w_H(\mathbf{e})+1}U_N(z) = \mu z^{d-w_H(\mathbf{e})+1}\sigma_{inv}(z).$$

Therefore, provided 0 is not an element of $\mathbf{x}$, $\hat{U}_d(z)$ allows to recover the error positions without ambiguity. Transposing this result to Patterson decoding requires to adapt both EEA's. The adaptation of the second one is straightforward. For the first one (syndrome inversion), a problem arises: the analogous of Proposition 3 would yield $\hat{U}_{N_I}(z) = \mu z^{k_i}(S^{-1}_{Gop,\mathbf{e}} \mod g)$ for some $k_i > 0$, and we found no way of determining when z is a factor of $S^{-1}_{Gop,\mathbf{e}} \mod g$. However, we can protect the second EEA to avoid the attack of 3.2.

Algorithm 8 EEA with regular flow

Input: $a(z) = z^{2t}, b(z) = S_{\mathbf{e}}(z), d = 2t$
Output: $\hat{U}_d(z) = \mu z^{d-w_H(\mathbf{e})+1}\sigma_{inv}(z)$,$\hat{R}_d(z) = \mu z^{d-w_H(\mathbf{e})+1}\omega_{\mathbf{e}}(z)$ for some $\mu \in \mathbb{F}_{q^m}^*$.

$\hat{R}_{-1}(z) \leftarrow a(z), \hat{R}_0(z) \leftarrow zb(z)$,
$\hat{U}_{-1}(z) \leftarrow 1, \hat{U}_0(z) \leftarrow 0$,
$\delta \leftarrow -1$.
for $j = 1, \ldots, d$ **do**
  $\alpha_j \leftarrow \hat{R}_{j-1,d}, \beta_j \leftarrow \hat{R}_{j-2,d}$.
  $temp_R(z) \leftarrow z\left(\alpha_j \hat{R}_{j-2}(z) - \beta_j \hat{R}_{j-1}(z)\right)$.
  $temp_U(z) \leftarrow z\left(\alpha_j \hat{U}_{j-2}(z) - \beta_j \hat{U}_{j-1}(z)\right)$.
  if $\alpha_j = 0$ (ie $\deg(\hat{R}_{j-1}) < \deg(\hat{R}_{j-2})$) **then**
    $\delta \leftarrow \delta + 1$.
  else
    $\delta \leftarrow \delta - 1$.
  end if
  if $\delta < 0$ **then**
    $(\hat{R}_j(z), \hat{R}_{j-1}(z)) \leftarrow (\hat{R}_{j-1}(z), temp_R)$
    $(\hat{U}_j(z), \hat{U}_{j-1}(z)) \leftarrow (\hat{U}_{j-1}(z), temp_U)$
    $\delta \leftarrow 0$.
  else
    $(\hat{R}_j(z), \hat{R}_{j-1}(z)) \leftarrow (temp_R, \hat{R}_{j-2}(z))$
    $(\hat{U}_j(z), \hat{U}_{j-1}(z)) \leftarrow (temp_U, \hat{U}_{j-2}(z))$
    $\delta \leftarrow \delta$.
  end if
end for
return $\hat{U}_d(z), \hat{R}_d(z)$

} L

5 Conclusion

We proposed an algorithm determining the error-locator polynomial costing always $16t^2$ field multiplications on any input. It contains a test depending on secret data, followed by two balanced branches. The indistinguishability of those branches by an attacker is crucial for the security of the decryption, and depends on the architecture of the implementation.

Acknowledgements. We are thankful to J.-C. Faugère, A. Gouget, L. Perret and anonymous reviewers for their comments in the preparation of this paper.

References

1. R. Avanzi, S. Hoerder, D. Page, and M. Tunstall. Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems. *J. Cryptographic Engineering*, 1(4):271–281, 2011.
2. E. Berlekamp, R. McEliece, and H. van Tilborg. On the Inherent Intractability of Certain Coding Problems. *Information Theory, IEEE Transactions on*, 24(3):384–386, May 1978.

3. E. Berlekamp, G. Seroussi, and T. Po. A Hypersystolic Reed–Solomon Decoder. In V. K. Bhargava, S. B. Wicker, IEEE Communications Society, and IEEE Information Theory Society, editors, *Reed-Solomon codes and their applications*, pages 205–241. IEEE Press Piscataway, NJ, 1994.
4. D. J. Bernstein, T. Chou, and P. Schwabe. McBits: Fast Constant-Time Code-Based Cryptography. In G. Bertoni and J. Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 250–272. Springer, 2013.
5. B. Biswas. *Aspects de mise en oeuvre de la cryptographie basée sur les codes*. These, Ecole Polytechnique X, Oct. 2010.
6. S. Heyse. Implementation of McEliece Based on Quasi-dyadic Goppa Codes for Embedded Devices. In B.-Y. Yang, editor, *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 143–162. Springer, 2011.
7. F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North–Holland, Amsterdam, fifth edition, 1986.
8. R. Overbeck and N. Sendrier. Code-based cryptography. In D. Bernstein, J. Buchmann, and E. Dahmen, editors, *Post-Quantum Cryptography*, pages 95–145. Springer Berlin Heidelberg, 2009.
9. N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.
10. D. Sarwate and N. Shanbhag. High-speed architectures for Reed-Solomon decoders. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 9(5):641–655, Oct 2001.
11. D. Sarwate and Z. Yan. Modified Euclidean algorithms for decoding Reed-Solomon codes. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 1398–1402, June 2009.
12. A. Shoufan, F. Strenzke, H. G. Molter, and M. Stöttinger. A timing attack against patterson algorithm in the mceliece pkc. In D. Lee and S. Hong, editors, *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 161–175. Springer, 2009.
13. A. Shoufan, T. Wink, H. Molter, S. Huss, and E. Kohnert. A Novel Cryptoprocessor Architecture for the McEliece Public-Key Cryptosystem. *Computers, IEEE Transactions on*, 59(11):1533–1546, Nov 2010.
14. F. Strenzke. A Smart Card Implementation of the McEliece PKC. In P. Samarati, M. Tunstall, J. Posegga, K. Markantonakis, and D. Sauveron, editors, *Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices*, volume 6033 of *Lecture Notes in Computer Science*, pages 47–59. Springer Berlin Heidelberg, 2010.
15. F. Strenzke. A Timing Attack Against the Secret Permutation in the Mceliece PKC. In *Proceedings of the Third International Conference on Post-Quantum Cryptography*, PQCrypto'10, pages 95–107, Berlin, Heidelberg, 2010. Springer-Verlag.
16. F. Strenzke. Timing Attacks against the Syndrome Inversion in Code-Based Cryptosystems. In *PQCrypto*, pages 217–230, 2013.
17. F. Strenzke, E. Tews, H. G. Molter, R. Overbeck, and A. Shoufan. Side Channels in the McEliece PKC. In *Proceedings of the 2Nd International Workshop on Post-Quantum Cryptography*, PQCrypto '08, pages 216–229, Berlin, Heidelberg, 2008. Springer-Verlag.
18. Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa. A method for solving key equation for decoding goppa codes. *Inf. and Control*, 27(1):87 – 99, 1975.
19. K. Thull and C. Yap. A Unified Approach to HGCD Algorithms for polynomials and integers, 1990.