INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 12(1):7-25, February 2017.

Improved Timing Attacks against the Secret Permutation in the McEliece PKC

D. Bucerzan, P.L. Cayrel, V. Dragoi, T. Richmond

Dominic Bucerzan*
Aurel Vlaicu University of Arad
Department of Mathematics and Computer Science
Romania, 310330 Arad, Elena Dragoi, 2
Corresponding author:dominic@bbcomputer.ro

Pierre-Louis Cayrel
Laboratoire Hubert Curien, UMR CNRS 5516,
Université de Lyon, Saint-Etienne, France
pierre.louis.cayrel@univ-st-etienne.fr

Vlad Dragoi
Laboratoire LITIS - EA 4108
Université de Rouen - UFR Sciences et Techniques,
76800 Saint Etienne du Rouvray, France
vlad.dragoi1@univ-rouen.fr

Tania Richmond
Laboratoire IMATH, EA 2134,
Avenue de l'Université , BP 20132,
83957 La Garde Cedex, France
tania.richmond@univ-tln.fr

Abstract: In this paper, we detail two side-channel attacks against the McEliece public-key cryptosystem. They are exploiting timing differences on the Patterson decoding algorithm in order to reveal one part of the secret key: the support permutation. The first one is improving two existing timing attacks and uses the correlation between two different steps of the decoding algorithm. This improvement can be deployed on all error-vectors with Hamming weight smaller than a quarter of the minimum distance of the code. The second attack targets the evaluation of the error locator polynomial and succeeds on several different decoding algorithms. We also give an appropriate countermeasure.
Keywords: communication systems, theory of error correcting codes, code-based cryptography, McEliece PKC, side-channel attacks, timing attack, extended Euclidean algorithm.

1 Introduction

In the history of cryptography public key schemes are quite recent. In the classic era both encryption and decryption algorithms are symmetric, that is why having access to the keys gives a total control on both encryption and decryption steps. These types of schemes are called symmetric cryptosystem and are still widely used.

Nonetheless several security properties can hardly be achieved with the use of symmetric cryptography. That is the main reason public key cryptography appeared. The first public key cryptosystem was invented by Diffie and Hellman [5]. But despite the fact that public key schemes are rather new, they are widely spread in practice. Moreover there are few constructions to be used in practice and they are all based on number theory problems, more exactly the hardness of factoring and discrete logarithm problem. But this is rather concerning since there is little theoretical support indicating that these problems are indeed hard. One of the main threats for these schemes is the arrival of the quantum computer. Peter Shor has shown that both computation of discrete logarithm and factoring problem can be done in polynomial time on a quantum machine [13].

In reaction to this thread, several solutions have been proposed, such as hash-based cryptography, code-based cryptography, lattice-based cryptography, and multivariate cryptography. The new facts concerning the post-quantum cryptography are well discussed in [2]. Code-based cryptosystems were introduced in 1978 by Robert J. McEliece [8]. But many variants were attacked and partially or totally broken. Up to now, none of the proposed variants seemed as

strong and secure as the original McEliece public-key cryptosystem (PKC) using Goppa codes. Structural attacks managed to reveal the secret key and totally break variants that used the generalized Reed-Solomon codes [15] or QC-LDPC codes [10] and many other variants.

As the Goppa codes still resist to structural attacks, they present a real interest in our approach. So we focus our attention on the cryptanalysis of the McEliece PKC using Goppa codes. More exactly on the side-channel attacks using time differences between two executions of the same task. The interest of timing attacks is both practical and theoretical: we avoid unsecured implementations and discover new attacks succeeding in a polynomial time. The main purpose of these type of attacks is to reveal a part of the secret key and a breaking point of an algorithm. The authors of such exploits usually end up by giving the necessary countermeasures and the secure variant of the algorithms.

In the case of the McEliece PKC using Goppa codes, most of the timing attacks were discovered since 2008. Falko Strenzke's articles mention several weak points mostly situated in the decoding algorithm [14, 16, 18, 19]. Some of these can be repaired by an intelligent and cautious way of the programming manner where countermeasures were proposed in [1, 3, 19]. All of the mentioned attacks were realised on a McEliece PKC implementation using the Patterson algorithm (cf. Fig. 1) for decoding Goppa codes. The number of error corrections in the Patterson algorithm is bounded: up to t errors can be corrected, where t is the degree of the Goppa polynomial.

Our contribution is to reveal a new timing attack against the error-locator polynomial (ELP) evaluation and to improve two existing attacks. In our new version of the two existing attacks combined, we detail how the relation between the two attacks is crucial in order to avoid eventual errors. The attacks are executed on the extended Euclidean algorithm (EEA) and exploit the number of iterations. As the authors mentioned, the initial attacks are limited and may not allow the total break of the permutation. This limit is situated in the number of equations detected by their attack. We will use a new relation between the number of iterations in the two steps in order to expand the system and to fully determine the secret permutation. We will also give a single countermeasure, which is efficient to all types of attacks exploiting the EEA in this particular manner.

The second contribution is in giving a new timing attack against the ELP evaluation. The importance of this new attack is that it operates on the polynomial evaluation, applied in several decoding algorithms as the Patterson algorithm, Berlekamp-Massey algorithm or any general decoder for alternant codes. We will show that this attacks succeeds on several variants of the polynomial evaluation.

2 Background

For all the necessary background on coding theory we address the reader to any book in this field, for example [7]. Nevertheless we give here the details concerning binary Goppa codes.

2.1 Goppa codes

Definitions and Properties

We will focus exclusively on binary Goppa codes in this paper, but it is easy to generalize our results to q-ary codes:
-Goppa polynomial: $g(x)$ is a polynomial over $\mathbb{F}_{2^m}[x]$ with $\deg(g) = t$.
-Goppa support: $\mathcal{L} = \{\alpha_0, \alpha_1, .., \alpha_{n-1}\}$ subset of $\mathbb{F}_{2^m}$ s.t. $g(\alpha_i) \neq 0$.
The syndrome polynomial associated to $c \in \mathbb{F}_2^n$: $\mathcal{S}_c(x) = \sum_{i=1}^{n} \frac{c_i}{x+\alpha_i}$.

Definition 1 (Binary Goppa code)**.** Given $g(x), \mathcal{L}$ and $\mathcal{S}_c(x)$ the binary Goppa code is defined as:

$$\Gamma(\mathcal{L}, g) = \{c \in \mathbb{F}_2^n \mid \mathcal{S}_c(x) \equiv 0 \mod g(x)\}.$$

Among the most important properties that a Goppa code satisfies we recall the followings:

Proposition 2. *A Goppa code $\Gamma(\mathcal{L}, g)$ is a linear code over $\mathbb{F}_2$. Its length is given by $n = |\mathcal{L}|$, its dimension is $k \geq n - mt$, where $t = \deg(g)$ and its minimun distance $d \geq t + 1$.*

The syndrome polynomial $\mathcal{S}_c(x)$ satisfies the following property:

$$\mathcal{S}_c(x) = \tfrac{\omega(x)}{\sigma(x)} \mod g(x),$$

where $\sigma(x) = \prod_{i=1}^{t}(x + a_i)$ is called the error locator polynomial (ELP) and the elements $\forall i \in \{1, \ldots, t\} a_i \in \mathcal{L}$ are the error positions.

Irreducible binary Goppa codes are defined by an irreducible Goppa polynomial g and admit the maximum length $n = 2^m$. We use this type of codes in the rest of the paper and adopt the following notations:

- For the permutation of the support elements:
  $\Pi(\mathcal{L}) = \mathcal{L}' = (\Pi(0), \Pi(1), \ldots, \Pi(\alpha_i), \ldots, \Pi(\alpha_{n-2}))$. where Π is an element of the symmetric group.
- Let $P(x)$ be a monic polynomial of degree t over $\mathbb{F}_{2^m}$ with t roots denoted a_i:

  $$P(x) = x^t + S_{t-1}^t x^{t-1} + S_{t-2}^t x^{t-2} + \ldots + S_2^t x^2 + S_1^t x + S_0^t,$$

  where the coefficients $S_i \in \mathbb{F}_2^m$ are the elementary symmetric functions:

  $S_{t-1}^t = \sum_{i=1}^{t} a_i$, $S_{t-2}^t = \sum_{\substack{i=1,j=1 \\ i \neq j}}^{t} a_i a_j$, $\ldots$,

  $S_1^t = \sum_{j=1}^{t} \prod_{\substack{i=1 \\ i \neq j}}^{t} a_i$ and $S_0^t = \prod_{i=1}^{t} a_i$.

Alternant decoders

For the (irreducible) binary Goppa codes, we can use (at least) three different decoding algorithms: 1. the extended Euclidean algorithm (EEA); 2. the Berlekamp-Massey algorithm; 3. the Patterson algorithm.

Extended Euclidean Algorithm (EEA). The first decoding algorithm can correct up to $\frac{t}{2}$ errors. We can increase the error-correction capabcity and correct up to t errors when the syndrome associated to g^2 is used. Unfortunately, the corresponding parity check matrix has two times more rows and the construction is more complex.

The Berlekamp-Massey algorithm. Similarly as the EEA, the Berlekamp-Massey algorithm has to use g^2 in order to decode t errors. The advantage of this algorithm is that it isn't vulnerable to several existing timing attacks and it allows a fast and constant-time computation. Some advantages are listed in [3].

The Patterson algorithm. The Patterson algorithm offers another solution for the syndrome decoding. The decryption described in [12] permits to correct up to t errors by using the syndrome associated to g but not to g^2.

2.2 The McEliece Cryptosystem

The McEliece PKC [8] is composed by the three following algorithms.

*Key generation:*The first step is to generate the support (the set of $n = 2^m$ elements) and the Goppa polynomial g of degree t. Then, the parity check matrix can be built and brought to a systematic form: $[\mathcal{I}_{n-k}|R]$ in order to recover a generator matrix G of the Goppa code. We randomly choose a non-singular $k \times k$ matrix S and a $n \times n$ permutation matrix Π, and compute the public $k \times n$ generator matrix $\mathcal{G} = SG\Pi$. The key generation procedure outputs $\mathsf{sk} = (\Gamma(\mathcal{L}, g), S, \Pi)$ and $\mathsf{pk} = (n, t, \mathcal{G})$.

Message encryption:

- *Inputs:* message $\mathsf{m} \in \mathbb{F}_2^k$, public key $\mathsf{pk} = (n, t, R^T)$.
- *Output:* ciphertext $\mathsf{z} \in \mathbb{F}_2^n$.

1. Randomly choose an n-bit error-vector with weight $\mathsf{wt}(e) = t$;
2. Encode $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$;
3. Return z.

Message decryption:

- *Inputs:* ciphertext $\mathsf{z} \in \mathbb{F}_2^n$, secret key $\mathsf{sk} = (\Gamma(\mathcal{L}, g), S, \Pi)$.
- *Output:* message $\mathsf{m} \in \mathbb{F}_2^k$.

1. Compute $\mathsf{z}' = \mathsf{z}\Pi^{-1}$;
2. Find $\mathsf{m}' = \mathsf{m}S$ from $\mathsf{z}' \oplus e$ using $\mathcal{D}ecode(\mathsf{z}')$ with the secret code;
3. Compute $\mathsf{m} = \mathsf{m}'S^{-1}$;
4. Return m.

$\mathcal{D}ecode(.)$ is an alternant decoder (presented in the previous subsection).

Existing side-channel attacks There are several papers on side-channel attacks against the McEliece PKC and a quick review must be done in order to clear up the reader's understanding. Most of the attacks target the Patterson decoding algorithm and exploit several weaknesses.

Table 1: Patterson algorithm: existing timing attacks and countermeasures

Step	Ref.	Countermeasure
❶ $\mathsf{z}' = \mathsf{z}\Pi^{-1}$		
❷ $\mathcal{S}_{\mathsf{z}'}(x) = \mathcal{H}'\mathsf{z}'(x^{t-1}, \ldots, x^2, x, 1)^T$		
❸ $\mathcal{S}_{\mathsf{z}'}(x)^{-1} \mod g(x)$ *via EEA*	[18]	control flow
❹ $\tau(x) = \sqrt{x + \mathcal{S}_{\mathsf{z}'}(x)^{-1}}$		
❺ $b(x)\tau(x) \equiv a(x) \mod g(x)$ $\deg(a) \leq \lfloor \frac{t}{2} \rfloor$; $\deg(b) \leq \lfloor \frac{t-1}{2} \rfloor$ *via EEA*	[14, 16]	in EEA make sure $\deg(r_i) = \deg(r_{i-1}) - 1$ and $\deg(\tau) = t - 1$
❻ $\sigma(x) = a^2(x) + xb^2(x)$		
❼ $e = (\sigma(\alpha_0), \sigma(\alpha_1), \ldots, \sigma(\alpha_{n-1})) \oplus (1, \ldots, 1)$	[1, 17, 19]	the non-support or make sure $\deg(\sigma) = t$
❽ $e' = e\Pi$		
❾ $\mathsf{z} = \mathsf{z}' \oplus e'$		

There are mainly two types of attacks classified by their goal:
1. Attacks recovering the secret message m [1, 17, 19]; 2. Attacks recovering (fully or partially) the secret key sk [14, 16–18].

The attacks on steps ❸ and ❺ are able to determine some relations on the support elements by counting the number of iterations in the EEA. We improve it in Section 3.

The attack on step ❼ reveals error positions using timing differences in the ELP evaluation. The attacker is able to find the error-vector with a certain non negligible probability. The basic idea is that two different polynomials, with some different degrees, are not evaluated in the same time. So the timing difference gives some information on the error-vector. We improve this attack in Section 4.

In the rest of this paper, we assume that an attacker chooses a weight $0 < r < t$ for the error-vector e and we use the following notations: $\deg(g) = t$ and $\mathsf{wt}(e) = r$.

In the next Sections we will detail the complexity analysis of these attacks as well as adapted parameter values.

3 Timing attack against double using of the EEA

Goal: The attacker's goal is to recover the secret permutation Π.

Identification of a leakage: The leakage is identified at steps ❸ and ❺ of the Patterson algorithm. This type of attack was already published in [16, 18]. The two steps using the EEA are considered as independent parts. In this section, we propose to show the relation existing between both steps and thus attack them. In fact, the main problem of previous attacks is the limited number of cases that can be exploited. They just can be applied on $\mathsf{wt}(e) \in \{2, 4\}$ as shown in [16] or $\mathsf{wt}(e) \in \{2, 4, 6\}$ as presented in [18].

The problem comes from a simple fact: the number of iterations is given by two conditions. One of the condition is that all of the quotients in the EEA must be polynomials of degree equal to one. So when this condition is not fulfilled the number of iterations could not be any longer controlled by the attacker. We will use $N_{❸}$ and $N_{❺}$ as notations for the number of iterations in the 3rd step (respectively 5th step) of the Patterson algorithm.

Motivations of our attack: We will show that using the relation between both steps will allow us to fully control the number of iterations. The other contribution is in finding the relation between both steps and in using it for building a larger set of equations. We will show that we are able to extend the limited equation number of the system up to $\mathsf{wt}(e) = \frac{\deg(g)}{2}$.

The main interest is that instead of finding only equations involving the permutation of 2, 4 or maybe 6 elements, we can extend it as much as necessary in order to discover the secret permutation.

In terms of complexity, instead of enumerating all possible permutations, i.e. $n!$ permutations, we reduce the complexity to the following expression:

$$\sum_{i=3}^{p} \binom{n}{i}, \quad \text{where } p \leq \frac{\deg(g)}{2} - 1$$

Hence for small values of p, we have:

$$\sum_{i=3}^{p} \binom{n}{i} \leq \binom{n}{p} \times (p-2) \leq \left(\frac{en}{p}\right)^{p} \times (p-2)$$

(where $e = 2.718281828\ldots$ is the basis of the natural logarithms).

In order to make clear the difference between the naive attack and our attack, we propose to give a lower bound for the complexity of the naive attack:

$$\left(\frac{n}{e}\right)^n \leq n!$$

Finally:

$$\sum_{i=3}^{p} \binom{n}{i} \leq \left(\frac{en}{p}\right)^p \times (p-2) << \left(\frac{n}{e}\right)^n \leq n!$$

So the naive attack would be exponential in the length of the code as a timing attack would only have a complexity exponential in the maximum of the error-vector's weight needed for the attack (often extremely small in comparison with the code's length).

Scenario: The attacker proceeds in the three following steps:

1. He chooses a random message m and computes $c = \mathsf{m}\mathcal{G}$;
2. He randomly chooses an error-vector e of small weight $\mathsf{wt}(e) < t$ (t is the correction capacity of the code) and computes $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$;
3. He sends z to an oracle ($\mathcal{O}$), which outputs the message m and the number of iterations in steps ❸ and ❺ of the Patterson algorithm from Fig. 1.

Main idea: For $\mathsf{wt}(e) = 2p$ with $p \in \mathbb{N}$, the attacker will find equations having the following form: $\sum_{i=1}^{\mathsf{wt}(e)} \Pi(\alpha_i) = 0$. He will be able to build this type of equations with $0 < \mathsf{wt}(e) < \frac{\deg(g)}{2}$. We will denote by $N_❸$ the number of iterations in the ❸ step.

Conditions: The general assumption is that the attacker knows the public key pk, the order of all elements in the support $\mathcal{L}$ ($\mathcal{L}$ is supposed to be public, for example in the lexicographic order) and has access to an oracle $\mathcal{O}$. These assumptions are the same as in previously mentionned works. We improve the attack in the same context. The oracle $\mathcal{O}$ is also able to give some extra informations: the timing for the whole particular algorithm or just one step. We assume that the attacker can violate the procedure by adding $\mathsf{wt}(e) < t$ errors. The attacker is able to choose the number and the positions of errors.

3.1 Step ❸ in the Patterson algorithm

It was shown in [18] that the syndrome inversion leaks some information. The attack is based on the number of iterations used in the EEA, in order to compute the inverse of the syndrome polynomial $S(x)$ modulo the Goppa polynomial $g(x)$. It uses the following properties:

$$N_❸ \leq \deg(\sigma) + \deg(\sigma'), \text{ for } \mathsf{wt}(e) < \frac{\deg(g)}{2}.$$

We will not detail here all conditions, as they are well explained in [18], but we only give some important facts in order to make things clearer and to prepare the attack. Let us consider the ELP

$$\sigma(x) = x^r + S^r_{r-1}x^{r-1} + S^r_{r-2}x^{r-2} + \cdots + S^r_2 x^2 + S^r_1 x + S^r_0,$$

with $r \equiv 0 \mod 2$. Then $\sigma'(x) = S^r_{r-1}x^{r-2} + \cdots + S^r_3 x^2 + S^r_1$.

In this case the maximum number of iterations is given by the coefficient $S^r_{r-1} = \sum_{i=1}^{r} a_i$. So if $S^r_{r-1} \neq 0$, we obtain $N_{❸} = 2r - 2$ and all quotients have a degree equal to 1. If $S^r_{r-1} = 0$, $S^r_{r-3} \neq 0$, then $N_{❸} = 2r - 4$ and all quotients have a degree equal to 1.

3.2 Step ❺ in the Patterson algorithm

Locate the leakage: Two observations have to be done in order to understand and to locate the leakage point. The first one is about the number of iterations. It was proven (in [14]) that this number is (with a high probability):

$$N_{❺} = \sum_{i=1}^{N_{❺}} \deg(q_i) = \deg(b).$$

In the following paragraph, we will give some relations between $\tau(x)$, $b(x)$, $a(x)$ and $\sigma(x)$ (given in steps ❹, ❺ and ❻ in Fig. 1). We will prove some new relations. The new relations between these polynomials allow us to build the attack in such a manner that previous ambiguous cases were eliminated. These relations are crucial for better understanding of the entire decryption algorithm as they influence each step of the process and each particular form of the involved polynomials.

There are some useful properties that are going to be used in our approach:

Proposition 3. *1. If* $r \equiv 0 \mod 2$, *then* $\deg(a) = \frac{r}{2}$ *see [14].*

2. If $r \equiv 1 \mod 2$, *then* $\deg(b) = \frac{r-1}{2}$ *see [14].*

3. If $\deg(\tau) \leq \lfloor \frac{r}{2} \rfloor$, *then* $\deg(a) = \deg(\tau) + \deg(b)$.

4. If $\deg(\tau) \leq \lfloor \frac{r}{2} \rfloor$ *and* $\deg(\tau) \neq 0$, *then* $\mathsf{wt}(e) \equiv 0 \mod 2$.

Fact:

- *When* $\mathsf{wt}(e)$ *is odd:* For an error-vector with Hamming weight $\mathsf{wt}(e) = 2k + 1$, with $k \leq p - 1$, we have the following relations:
  $\deg(b) = k$, $\deg(a) \leq k$ and $\deg(\tau) \geq 2p - k$.
- *When* $\mathsf{wt}(e)$ *is even:* For an error-vector with Hamming weight $\mathsf{wt}(e) = 2k$, with $k \leq p$, we have the following relations:
  $\deg(a) = k$, $\deg(b) \leq k - 1$ and $\deg(b) = 0 \Leftrightarrow \deg(\tau) = k$.

3.3 Number of iterations

We saw that the number of iterations in the EEA equals $\deg(b)$ so we will focus on the form of the polynomial $b(x)$. More exactly, in the case when r is even. Let:

$$\sigma(x) = x^{2p} + S^{2p}_{2p-1}x^{2p-1} + S^{2p}_{2p-2}x^{2p-2} + S^{2p}_{2p-3}x^{2p-3} + \ldots + S^{2p}_{2}x^2 + S^{2p}_{1}x + S^{2p}_{0}.$$

We separate odd powers from even ones and get:

$$\begin{aligned}
\sigma(x) &= (x^{2p} + S^{2p}_{2p-2}x^{2p-2} + \ldots + S^{2p}_{2}x^2 + S^{2p}_{0}) \\
&\quad +(S^{2p}_{2p-1}x^{2p-1} + S^{2p}_{2p-3}x^{2p-3} + \ldots + S^{2p}_{1}x) \\
\sigma(x) &= (x^p + \sqrt{S^{2p}_{2p-2}}x^{p-1} + \ldots + \sqrt{S^{2p}_{2}}x + \sqrt{S^{2p}_{0}})^2 \\
&\quad +x(\underbrace{\sqrt{S^{2p}_{2p-1}}x^{p-1} + \ldots + \sqrt{S^{2p}_{1}}}_{b(x)})^2
\end{aligned}$$

So $\deg(b)$ is given by the coefficients S^{2p}_{2i-1} with $i \in \{1, 2, \ldots, p\}$. Therefore the number of iterations could be given by the same coefficients under an extra condition: all of the quotients have a degree equal to one. So we can distinguish $p-1$ possible cases depending on the coefficients, if the degree of the coefficients is equal to 1 in each iteration. Therefore:

$$\begin{cases} N_{\mathbf{5}} = p-1 & \text{if } S^{2p}_{2p-1} \neq 0 \\ N_{\mathbf{5}} = p-2 & \text{if } S^{2p}_{2p-1} = 0 \text{ and } S^{2p}_{2p-3} \neq 0 \\ N_{\mathbf{5}} = p-3 & \text{if } S^{2p}_{2p-1} = 0\ S^{2p}_{2p-3} = 0 \text{ and } S^{2p}_{2p-5} \neq 0 \\ \quad \vdots & \end{cases}$$

In all cases, the same assumption is made: the degree of the quotient equals 1 in each iteration. It means that we might have the number of iterations without any condition on the coefficients.

3.4 Attack against the pair $(N_{\mathbf{3}}, N_{\mathbf{5}})$

How it works. In this paragraph, we will explain how our attack works. We start by presenting the general relation for the pair $(N_{\mathbf{3}}, N_{\mathbf{5}})$. Using 3.1 and 3.2 we get the following property:

Proposition 4. *Let* $\mathsf{wt}(e) = 2p < t/2$.

$$(N_{\mathbf{3}}, N_{\mathbf{5}}) = (4p-4, p-2) \Rightarrow \sum_{i=1}^{2p} a_i = 0$$

with probability $\mathcal{P}_{success}$.

We will give a snapshot of each step in the attack and give some information on the success probability.

1. Find the position of $\Pi(0)$ (see [14]).
2. Set $\mathsf{wt}(e) = 4$:
   Fix $\Pi(0) \in \{\text{error-vector}\}$ and find $\sum_{i=1}^{3} a_i$ with $a_i \neq 0$.
   Fix $\Pi(0) \notin \{\text{error-vector}\}$ and find $\sum_{i=1}^{4} a_i$ with $a_i \neq 0$.
3. Set $\mathsf{wt}(e) = 6$:
   Fix $\Pi(0) \in \{\text{error-vector}\}$ and find $\sum_{i=1}^{5} a_i$ with $a_i \neq 0$.
   Fix $\Pi(0) \notin \{\text{error-vector}\}$ and find $\sum_{i=1}^{6} a_i$ with $a_i \neq 0$.
4. ...
5. Set $\mathsf{wt}(e) = \lfloor \frac{t}{2} \rfloor$:
   Fix $\Pi(0) \in \{\text{error-vector}\}$ and find $\sum_{i=1}^{\mathsf{wt}(e)-1} a_i$ with $a_i \neq 0$.
   Fix $\Pi(0) \notin \{\text{error-vector}\}$ and find $\sum_{i=1}^{\mathsf{wt}(e)} a_i$ with $a_i \neq 0$.

In Appendix 1, a toy example is presented for a better comprehension.

3.5 Success probability

The success probability $\mathcal{P}_{success}$ is described by the following event: *{All the quotients have a degree=1}*. If we consider all elements of our support as uniformly distributed variables and the independence of each step inside the EEA, under the initial assumptions we have:

$$\begin{aligned}\mathcal{P}_{success} &= \mathcal{P}(\{N_{❸} = 4p-4\} \cap \{N_{❺} = p-2\}) \\ &= \mathcal{P}(\{N_{❸} = 4p-4\})\mathcal{P}(\{N_{❺} = p-2\}) \\ &= (1-\tfrac{1}{n})^{N_{❸}+N_{❺}}.\end{aligned}$$

Experimental results show that for $n = 2048$ and $\mathsf{wt}(e) = 4$, in order to find equations of the following form: $\Pi(\alpha_1) + \Pi(\alpha_2) + \Pi(\alpha_3) + \Pi(\alpha_4) = 0$ the probability equals 0.998. It means that less that 0.2% of the cases are not exploitable among all possible cases under the condition: $N_{❸} = 4$ and $N_{❺} = 0$. In other words, each time this combination is revealed, the probability of having a good equation for our attack equals 0.998 for the given parameters.

3.6 Experimental work

In order to validate the relations that we presented in the previous paragraph for $(N_{❸}, N_{❺})$, we used a *Pari/GP* implementation of the McEliece cryptosystem (the code will be publicly available). We computed a keypair, then encoded and decoded a given message multiple times, by checking the value of the couple $(N_{❸}, N_{❺})$, searching for the valid combinations described above. We also used different values for m, the extension degree of the finite field. We ran the algorithm until we got the specific combination about hundred times. Then, we obtained an average value for the necessary iterations required to get the searched combination. The results are presented in the following table:

Table 2: Number of necessary iterations to get the combination for error-vectors of Hamming weight 4, 6 and 8

Combination:			
$N_{❸}$	4	8	12
$N_{❺}$	0	1	2
Number of iterations for $m = 7$	127	138	142
Number of iterations for $m = 8$	235	270	273

It means that for $m = 7$, we need to send to the oracle in average 127 different ciphertexts, in order to get the wanted relation $(\Pi(\alpha_1) + \Pi(\alpha_2) + \Pi(\alpha_3) + \Pi(\alpha_4) = 0)$. In the case of the previous equation, the density of such configurations equals in average

$$\frac{1}{127} \times \mathcal{P}_{success} = \frac{1}{127} \times \left(1 - \frac{127}{128}\right)^4.$$

Knowing one relation allows us, by fixing one of the positions, to reduce the number of ciphertexts that has to be sent to the oracle. It means that wanted relations are revealed more often as we progress in the attack. It also gives the first intuition on the structure of the permutation (see Appendix 1).

Attack implementation In order to practically test our attack, we used the same software implementation. In order to reveal timings close to real values, we repeated the attack for more than 10^6 times. We presented the obtained results are in the following table:

Table 3: Timings (in sec.) for decryption in the case of $n = 2^{11}$ and $t = 16$

$\mathsf{wt}(e)$	Timings for expected attack equation	Timings for random type equation
4	30141892×10^{-6}	304856×10^{-4}
6	3072799×10^{-5}	310234×10^{-4}
8	31597171×10^{-6}	32242382×10^{-6}
10	3285724×10^{-6}	3345847×10^{-5}

Remarks: We didn't give the timings for the odd values as they are constant and independant from the linear combinations between the permutations of the error positions. From Figure 3, we observe that there's a slight difference between the attack on this type combinations and on randomly distributed combinations. As we mentioned before for the random combinations those with the maximum number of iterations are more likely to appear (the case when all coefficients are different from zero). So in this case, we have the timing difference required for our attack to succeed. In Section 3.7, we will explain how the patch will work not only on this type of attacks but even on other types as the bit-flipping attacks.

3.7 Countermeasures

We have seen that it is possible to attack a system by knowing how many times the EEA is repeated. The number of iterations can go from 0 to $t-1$ in the syndrome inversion and from 0 to $t/2$ in the ELP determination. In order to avoid a correlation-finding from the number of iterations, we propose to introduce extra iterations into the EEA. The number of extra iterations should be chosen between 0 and a value that we call *extra*. The *extra* value is either $t/2$ or $t-1$, for the syndrome inversion ❸ or the ELP determination ❺, respectively. The variable i contains the number of iterations realized in the first part of the secured EEA. We chose to use integer values in the extra EEA steps, in order to avoid divisions by zero that may occur if we keep the previous terms. The point is to keep computing things that are as computationally expensive as the original EEA, so that an attacker can't make the difference between true steps and extra steps. We present the proposed secured modified EEA:

Input: $f(x)$, $g(x)$, d_{break} and t.
Output: $a(x)$ and $b(x)$ s.t. $a(x) \equiv b(x)f(x) \mod g(x)$

1. $d \leftarrow d_{break}$
2. $[b_{-1}, b_0] \leftarrow [0, 1]$
3. $[r_{-1}, r_0] \leftarrow [g(x), f(x)]$
4. $i \leftarrow 0$
5. While $\deg(r_i) > d$ do

$$\begin{array}{rcl} i & \leftarrow & i+1 \\ r_{i-2}(x) & = & r_{i-1}(x)q_i(x) + r_i(x) \\ b_i(x) & \leftarrow & b_{i-2}(x) + q_i(x)b_{i-1}(x) \end{array}$$

   end while

6. $a(x) \leftarrow r_i(x)$
7. $b(x) \leftarrow b_i(x)$

8. $extra = f(t, d_{break})$
9. While $i < extra$ do

$$\begin{array}{rcl} i & \leftarrow & i+1 \\ r_{i-2}(x) & = & 3q_i(x)+5 \\ b_i(x) & \leftarrow & 5+6q_i(x) \end{array}$$

end while

The new security parameters: We recall the fact that normal security parameters do not take in consideration timing attacks. Usually security parameters are given under the assumption of possible naive attacks or known structural attacks as ISD [9] For example for the McEliece PKC the usual parameters are:

$$\begin{array}{c} \text{100-bit security } n = 2048,\ t = 50 \\ \text{128-bit security } n = 2960,\ t = 56 \\ \text{256-bit security } n = 6624,\ t = 115 \end{array}$$

For the first parameters a timing attack with $p = 6$ would reveal a complexity less than 2^{61} elementary operations, that is way lower than the original security level proposals. So for timing attacks larger parameters have to be taken in consideration in order to maintain the same level of security. For example in order to same a 100 bit security lever against this type of timing attacks one should propose $n = 131072$.

The usual solution is not to increase the values of the parameters but to propose secure variant of the algorithm, variant that is not vulnerable to the specified attack. Our proposal is less faster that the original algorithm, it operates $(t-1) \times O(1)$ for the syndrome inversion and $\frac{t}{2} \times O(1)$ for the key equation (where $O(1)$ is the usual complexity for a division).

Meanwhile it is secure against timing attacks described below. The proof is very simple and it based on the fact that this particular type of timing attacks are based on the number of iterations is the EE Algorithm. Since our algorithm performs the same number of iterations no matter what relations are hidden between the polynomial coefficients it can't reveal any of such secret relations.

Once the countermeasure was applied, we ran the same attack and got the following timings for selected Hamming weights (the average timings are presented for more than 10^7 simulations in Fig. 4).

Table 4: Timings (in sec.) for decryption in the case of $n = 2^{11}$ and $t = 10$

$\mathsf{wt}(e)$	Timings for attack type equations	Timings for a random type combination
6	57.99	57.90
7	57.89	
8	57.94	58.03
9	58.33	
10	57.81	57.89

Remark: We observe that the protected implementation is impossible to attack (using the same techniques). We stress that the proposed countermeasure is also efficient in the case when an attacker wants to use previous techniques, like in [14, 16, 18].

4 Timing attack against the ELP evaluation

Goal: The attacker's goal is to find the secret permutation Π.

Identification of a leakage: A leakage is identified at step ❼ of the Patterson algorithm: the ELP evaluation. We recall that the ELP is denoted σ in Subsection 2.1. The attack is based on the fact that the form of the polynomial differs from the element to decode. We will prove that the algorithm's complexity is strongly related to the coefficients of $\sigma(x)$. We will then perform a timing attack on the ELP evaluation and control the values of the coefficients of $\sigma(x)$.

Motivations of our attack: One of the main motivations of our attack is that it can operate on all existing implementations of a general alternant decoder. It operates on the ELP evaluation, step that has to be computed in any decoding algorithm.

We will give two basic algorithms for the ELP evaluation with some improvements and show that even with the published improvements our attack succeeds. We will choose the polynomial evaluation from right to left (the naive algorithm) and from left to right (the Ruffini-Horner scheme). Imagine that our polynomial has a degree equal to an integer t. The first algorithm computes the result within $3t-1$ operations (t additions and $2t-1$ multiplications). As for the second one it computes the result within $2t$ operations (t additions and t multiplications). It was proven by V. Pan in 1966 [11] that the Ruffini-Horner's scheme [6] is optimal in terms of complexity.

The main idea of the improvement is to use the fact that some support elements have particular properties (like 0 and 1). Knowing the fact that one coefficient equals zero fasten up the algorithm as operations like multiplication or sum have fix values if they take zero as one of the input element. The same thing happens within the multiplication by one. So we will exploit these properties in order to improve our implementation. Each time a coefficient equals one or zero it will be store in a special table used afterwards for multiplication or addition. The case where a coefficient equals zero is rare and its probability has been studied in [4].

Nevertheless, each time there's a coefficient equal to zero we will no longer multiply it by the corresponding element as the multiplication equals zero. So we will use the predefined tables to get rid of the useless operations. We will proceed exactly the same way when the multiplication of an element has to be done when a coefficient equals to one. So each time we have one coefficient equal to zero, using our predefined tables, we get rid of two operations (one addition and one multiplication).

Scenario: The attack scenario is the same as in the previous attack except for the last step. In fact, the attacker gets the running time for the ELP evaluation in this section (step ❼ in Figure 1).

Idea: For $\mathsf{wt}(e)=2$, the attacker will find the positions of $\Pi(0)$ and $\Pi(1)$ the permutation of zero and one. After enough iterations, he will fix those two positions and repeat this attack with $\mathsf{wt}(e)=3$, he will then find the secret permutation Π (using exhaustive search for the remaining positions).

Conditions: The assumptions are the same as in the previous attack excepted that the attacker does not know the order of the elements in the support $\mathcal{L}$.

4.1 Success probability

As we said, in this attack we will only consider polynomials with a degree lower than three. For the case $r = 3$ we will give the full table of probabilities. We will start with the following general problem:

Problem: Let $P(x)$ be a monic polynomial of degree r with r distinct roots over $\mathbb{F}_{2^m}$. What is the probability that all its coefficients are different from zero?

This problem was treated in [4] and the results show that the probability can be bounded. For the classical parameters of the McEliece PKC, i.e. $n = 2048$ and $t \leq 50$, the authors obtain:

$$\mathcal{P} \geq 0.95$$

Proposition 5. *Let $P(x)$ be a monic polynomial of degree 3 with three distinct roots over $\mathbb{F}_{2^m}$ and $m \mod 2 = 1$.*

The probability $\mathcal{P}_3$ that all its coefficients are different from zero satisfies:

$$\mathcal{P}_3 = 1 - \frac{5}{2^m}.$$

4.2 Finding the permutation of the support elements zero and one

1. Consider the error-vectors e_i with $\mathsf{wt}(e_i) = 1$.

   In this case, the error locator polynomial has the following form:

   $$\sigma(x) = x + a_i, \text{ with } a_i \in \mathcal{L} = \{0, 1, \alpha, \ldots, \alpha^{n-2}\}.$$

   If $a_i \neq 0$, there is one addition $(+)$ in the $\sigma(x)$ evaluation.

2. Consider the error-vectors e_i with $\mathsf{wt}(e_i) = 2$.

   In this case, the error locator polynomial has the following form:

   $$\sigma(x) = x^2 + S_1^2 x + S_0^2, \text{ with } S_1^2 = a_i + a_j \text{ and } S_0^2 = a_i a_j.$$

   We distinguish two possible cases:

   (a) $\sigma(x) = x^2 + S_1^2 x + S_0^2$ if $a_i a_j \neq 0$

   (b) $\sigma(x) = x^2 + S_1^2 x$ if $a_i a_j = 0$

   The case (b) leads to a computation of the polynomial evaluation with one extra addition $(+)$ and the timings reveal all the couples $(\alpha_i, 0)$. We can assume now that the position of $\Pi(0)$ is known.

3. We fix this position and we seek for the position of $\Pi(1)$. Since the polynomial $\sigma(x) = x^2 + S_1^2 x$, the fastest evaluation is obtained for the couple $(\Pi(0), \Pi(1))$ as there is only one addition $(+)$ and one square computation.

4.3 Attack scenario when $r = 3$

We will consider error-vectors with Hamming weight that equals 3. The corresponding $\sigma(x)$ polynomial has always one of the eight following representations:

1. $\sigma(x) = x^3 + S_2^3 x^2 + S_1^3 x + S_0^3$ if $S_1^3 S_2^3 S_0^3 \neq 0$
2. $\sigma(x) = x^3 + S_2^3 x^2 + S_1^3 x$ if $S_0^3 = 0$ and $S_2^3 S_1^3 \neq 0$
3. $\sigma(x) = x^3 + S_2^3 x^2 + S_0^3$ if $S_1^3 = 0$ and $S_2^3 S_0^3 \neq 0$
4. $\sigma(x) = x^3 + S_1^3 x + S_0^3$ if $S_2^3 = 0$ and $S_1^3 S_0^3 \neq 0$
5. $\sigma(x) = x^3 + S_2^3 x^2$ if $S_2^3 \neq 0$ and $S_1^3 = 0$ and $S_0^3 = 0$
6. $\sigma(x) = x^3 + S_1^3 x$ if $S_1^3 \neq 0$ and $S_2^3 = 0$ and $S_0^3 = 0$
7. $\sigma(x) = x^3 + S_0^3$ if $S_0^3 \neq 0$ and $S_2^3 = 0$ and $S_1^3 = 0$
8. $\sigma(x) = x^3$ if $S_0^3 = 0$ and $S_2^3 = 0$ and $S_1^3 = 0$

Straightforward we deduce the following cases:

$$
\begin{array}{ll}
(a).\ \sigma(x) = x^3 + S_2^3 x^2 + S_1^3 x + S_0^3 & \text{if } S_1^3 S_2^3 S_0^3 \neq 0 \text{ and } \mathcal{P} = \frac{n-5}{n} \\
(b).\ \sigma(x) = x^3 + S_2^3 x^2 + S_1^3 x & \text{if } S_0^3 = 0 \text{ and } S_1^3 S_2^3 \neq 0 \text{ and } \mathcal{P} = \frac{3}{n} \\
(c).\ \sigma(x) = x^3 + S_2^3 x^2 + S_0^3 & \text{if } S_1^3 = 0 \text{ and } S_1^3 S_0^3 \neq 0 \text{ and } \mathcal{P} = \frac{1}{n} \\
(d).\ \sigma(x) = x^3 + S_1^3 x + S_0^3 & \text{if } S_2^3 = 0 \text{ and } S_1^3 S_0^3 \neq 0 \text{ and } \mathcal{P} = \frac{1}{n}
\end{array}
$$

Several cases can be eliminated by considering the fact that we accomplished the first step and we know the position of $\Pi(0)$. If we consider all the error-vectors where $a_i \neq 0 \quad \forall i \in \{1, 2, \ldots, n-1\}$ (i.e. 0 is not a root of $P(x)$), we reduce the possibilities for $\sigma(x)$. The new form of the system is the following:

$$
\begin{cases}
\sigma(x) = x^3 + S_2^3 x^2 + S_1^3 x + S_0^3 & \text{if } S_1^3 S_2^3 S_0^3 \neq 0 \\
\sigma(x) = x^3 + S_2^3 x^2 + S_0^3 & \text{if } S_1^3 = 0 \text{ and } S_2^3 S_0^3 \neq 0 \\
\sigma(x) = x^3 + S_1^3 x + S_0^3 & \text{if } S_2^3 = 0 \text{ and } S_0^3 S_1^3 \neq 0
\end{cases}
$$

In all cases, x^3 must be computed so we will not consider this part in the timing differences. In the structure that computes the polynomial evaluation the fastest is the last one. But this case is performed only when $S_2^3 = 0$.

4.4 Finding the positions of two elements such that $\Pi(\alpha_j)\Pi(\alpha_k) = 1$

In order to increase the number of equations in our system, we exploit the fact that $(\mathbb{F}_{2^m})^*$ is cyclic.

Recall: we know the positions of $\Pi(0), \Pi(1)$ and $\Pi(\alpha_1) + \Pi(\alpha_2) + \Pi(\alpha_3) = 0$. Without loss of generality, we choose to fix "$\Pi(0)$" on the first position and choose two other positions such that the sum is different from 1.

We are able to do that because we know the position of "$\Pi(1)$" and the couples (α_1, α_2) such that $1 + \alpha_1 + \alpha_2 = 0$. We get two new positions b_1 and b_2 such that $b_1 + b_2 \neq 1$. The error locator polynomial is: $\sigma(x) = x^3 + S_2^3 x^2 + S_1^3 x$.

For $b_1 b_2 = 1$ we get $\sigma(x) = x^3 + S_2^3 x^2 + x$. This form is the fastest to be computed as there is one less multiplication compare to the other case.

4.5 System resolution

Number of equations:

We will give the number of linear and quadratic equations obtained by the attacker. Finding the positions of $\Pi(0)$ and $\Pi(1)$ reduces the search set to $(n-2)$ elements.

- The first set of linear equations:

$$\text{Equation type (1): } \Pi(\alpha_j)\Pi(\alpha_k) = 1 \Rightarrow \sharp eq. = \tfrac{n-2}{2}$$

The last equation is determined by all the other ones because for the last couple only one possible solution remains available. For instance, if the attacker finds $(\frac{n-2}{2} - 1)$ different equations the last equation can be directly determined.

- The second set of linear equations:

$$\text{Equation type (2): } \Pi(\alpha_j) + \Pi(\alpha_k) = 1 \Rightarrow \sharp eq. = \tfrac{n-2}{2}.$$

As the first set, the last one can be determined by all others. This comes from the fact that for the three positions, we fixed the position of $\Pi(1)$ as the first one. So we have $(n-2)$ possibilities on the second position. But there are two repetitions for each $(\Pi(1), \Pi(\alpha_j), \Pi(\alpha_k))$-vector.

- The third set of quadratic equations:

$$\text{Equation type (3): } \Pi(\alpha_i) + \Pi(\alpha_j) + \Pi(\alpha_k) = 0 \Rightarrow \sharp eq. = \tfrac{(n-2)(n-4)}{6}$$

The total number of equations for $\Pi(\alpha_i) + \Pi(\alpha_j) + \Pi(\alpha_k) = 0$ including the second set equals $(n-1)(n-2)$ as the third position is fixed and the two others are free and different. Here, the number of repetitions equals six. So we obtain $\left(\frac{(n-2)(n-1)}{6} - \frac{n-2}{2}\right)$ equations.
To illustrate how the attack works a toy example is given in Appendix 2.

Conclusion

In this article, we focused our attention on the cryptanalysis of the McEliece PKC with the binary Goppa codes. We showed the existing weak points in the Patterson decoding algorithm and determined the relations between the number of iterations in two different steps of the algorithm and the secret permutation. Since those relations were the main connection idea between the two extended Euclidean algorithms, we set up a timing attack based on this fact. The advantage of this attack is that it increased the probability of success by avoiding ambiguous cases, undetectable in previous attacks. The other advantage is that it allows higher expansion of the number of equations determined by the attacker in order to find the secret permutation.

The second important contribution of our article is a new attack that can be performed on several different decoding algorithms. It reveals that even intelligent variants of some polynomial evaluation algorithms might leak information and need to be patched or replaced. The ideas discovered in the attacks might be reused in any further implementations using the algorithms mentioned before. So secure variants must be used in order to avoid any leakage point.

Bibliography

[1] Roberto Avanzi, Simon Hoerder, Dan Page, Mike Tunstall (2010), Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems, *Cryptology ePrint Arch.*, Report 2010/479, 2010.

[2] Daniel J. Bernstein, Johannes Buchmann, Erik Dahmen (eds.) (2009), *Post-Quantum Cryptography*, Springer, 2009.

[3] Daniel J. Bernstein, Tung Chou, Peter Schwabe (2013), McBits: fast constant-time code-based cryptography, https://binary.cr.yp.to/mcbits-20130616.pdf, 1-26.

[4] Vlad Dragoi, Pierre-Louis Cayrel, Brice Colombier, Tania Richmond (2013), Polynomial structures in code-based cryptography, *Indocrypt 2013*, LNCS2850: 286-296.

[5] Whitfield Diffie, Martin Hellman (1976), New directions in cryptography, *IEEE Trans. Inform. Theory*, 22(6):644–654.

[6] W.G. Horner (1819), A new method of solving numerical equations of all orders by continuous approximation, *Phil. Trans. R. Soc. Lond.*, 109:308–335.

[7] Florence J. MacWilliams, Neil J. A. Sloane (1986), *The Theory of Error-Correcting Codes*, North–Holland, Amsterdam, 5th ed., 1986.

[8] Robert J. McEliece (1978), A public-key cryptosystem based on algebraic coding theory, *Jet Propulsion Laboratory DSN Progress*, Report 42-44, 114–116.

[9] Robert Niebuhr et al. (2010), On lower bounds for information set decoding over fq, In *C. Cid, J.-C. Faugere, (eds.), Proc. of the Second Intl. Conf. on Symbolic Computation and Cryptography, SCC 2010*, 143–157.

[10] Ayoub Otmani, Jean-Pierre Tillich, Leonard Dallot (2008), Cryptanalysis of a McEliece cryptosystem based on quasi-cyclic LDPC codes, *Proc. of First Intl. Conf. on Symbolic Computation and Cryptography (SCC 2008)*, 69–81.

[11] Victor Y. Pan (1966), On Methods of Computing the Values of Polynomials, *UspeKhi Mathematicheskikh Nauk*, 21:103–134.

[12] Nicholas J. Patterson (1975), The algebraic decoding of goppa codes, *IEEE Transactions on Information Theory*, 21(2): 203–207.

[13] Peter W. Shor (1997), Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Journal on Computing*, 26(5):1484-1509.

[14] Abdulhadi Shoufan et al. (2010), A Timing Attack against Patterson Algorithm in the McEliece PKC, *ICISC 2009*, LNCS 5984: 161–175.

[15] V.M. Sidelnikov and S.O. Shestakov (1992), On the insecurity of cryptosystems based on generalized Reed-Solomon codes, *Discrete Math. Appl.*, 2(4):439–444.

[16] Falko Strenzke (2010), A Timing Attack against the Secret Permutation in the McEliece PKC, In *N. Sendrier (ed.), Post-Quantum Cryptography, Third intl. workshop*, LNCS6061: 95–107.

[17] Falko Strenzke (2010), Fast and secure root-finding for code-based cryptosystems, *Cryptology ePrint Arch.*, Report 2011/672, 2011.

[18] Falko Strenzke (2011), Timing attacks against the syndrome inversion in code-based cryptosystems, *Cryptology ePrint Arch.*, Report 2011/683, 2011.

[19] Falko Strenzke et al. (2008), Side channels in the McEliece PKC, In *J. Buchmann and J. Ding (eds.), Post-Quantum Cryptography, Second intl. workshop*, LNCS5299: 216–229.

Appendix

1 Toy example for the EEA attack

Consider $\mathbb{F}_{2^4}[x] = \frac{\mathbb{F}_2[x]}{x^4+x+1}$. The generator matrix $\mathcal{G}$ of the Goppa code and the support $\mathcal{L} = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{14}\}$ are public. Let $\mathsf{m} \in \mathbb{F}_2^k$ be the message and $\mathcal{O}$ the decoding oracle. We notice that if $\mathcal{L}$ is public, one can find $G(x)$ such that $\mathcal{L} = \frac{\mathbb{F}_2[x]}{G(x)}$. The other way is equaly true: if $G(x)$ is public then one can easily find $\mathcal{L}$. Suppose that the secret permutation is:

$$\Pi(\mathcal{L}) = \mathcal{L}' = \{\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{14}, 0, 1\} = \{\ell_i \,|i \in (1 \ldots 16)\}$$

- 1^{st} *step:*
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 1$.
  - ⋆ $N_{❸}$ and $N_{❺}$ reveals the position of $\Pi(0)$: ℓ_{15}.
  - ◦ This is mainly due to: $\sigma(x) = x$ we have $\tau(x) = 0$ and $S^{-1}(x) = x$
- 2^{nd} *step:*
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 4$ (the positions $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3})$ are the three non-zero positions of e and $\ell_{i_4} = \ell_{15}$).
  - ⋆ The couple $\begin{pmatrix} N_{❸} \\ N_{❺} \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$ reveals all $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3})$ such that $\ell_{i_1} + \ell_{i_2} + \ell_{i_3} = 0$.
    Here $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}) \in \{(\ell_1, \ell_4, \ell_{16}), (\ell_3, \ell_{14}, \ell_{16}), \ldots\}$.
  - ◦ $\deg(\sigma) = 4$ and $\deg(\omega) = \begin{cases} 2 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} \neq 0 \\ 0 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} = 0 \end{cases}$
  - ◦ $\deg(b) = \begin{cases} 1 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} \neq 0 \\ 0 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} = 0 \end{cases}$
- 3^{rd} *step:*
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 4$ (the positions $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4})$ are the four non-zero positions of e).
  - ⋆ The couple $\begin{pmatrix} N_{❸} \\ N_{❺} \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$ reveals all $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4})$ such that $\ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \ell_{i_4} = 0$.
    Here $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}) \in \{(\ell_1, \ell_2, \ell_{10}, \ell_{16}), (\ell_2, \ell_3, \ell_{13}, \ell_{16}), \ldots\}$.
  - ◦ $\deg(\sigma) = 4$ and $\deg(\omega) = \begin{cases} 2 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \ell_{i_4} \neq 0 \\ 0 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \ell_{i_4} = 0 \end{cases}$
  - ◦ $\deg(b) = \begin{cases} 1 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \ell_{i_4} \neq 0 \\ 0 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \ell_{i_4} = 0 \end{cases}$
- 4^{th} *step:*
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 6$ (the positions $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}, \ell_{i_5})$ are the five non-zero positions of e and $\ell_{i_6} = \ell_{15}$).
  - ⋆ The couple $\begin{pmatrix} N_{❸} \\ N_{❺} \end{pmatrix} = \begin{pmatrix} 8 \\ 1 \end{pmatrix}$ reveals all $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}, \ell_{i_5})$ such that $\ell_{i_1} + \ell_{i_2} + \cdots + \ell_{i_5} = 0$.
    Here $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}, \ell_{i_5}) \in \{(\ell_1, \ell_2, \ell_3, \ell_{12}, \ell_{16}), (\ell_3, \ell_4, \ell_8, \ell_{12}, \ell_{16}), \ldots\}$.

$\circ$ $\deg(\sigma) = 4$ and $\deg(\omega) = \begin{cases} 4 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \ell_{i_4} + \ell_{i_5} \neq 0 \\ 2 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \ell_{i_4} + \ell_{i_5} = 0 \end{cases}$

$\circ$ $\deg(b) = \begin{cases} 2 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \ell_{i_4} + \ell_{i_5} \neq 0 \\ 1 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \ell_{i_4} + \ell_{i_5} = 0 \end{cases}$

- 5^{th} *step:*
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 6$ (the positions $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ell_{i_4}, \ell_{i_5}, \ell_{i_6})$ are the six non-zero positions of e).
  - $\star$ The couple $\binom{N_{\textcircled{8}}}{N_{\textcircled{5}}} = \binom{8}{1}$ reveals all $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ldots, \ell_{i_6})$ such that $\ell_{i_1} + \ell_{i_2} + \cdots + \ell_{i_6} = 0$.
    Here $(\ell_{i_1}, \ell_{i_2}, \ell_{i_3}, \ldots, \ell_{i_6}) \in \{(\ell_1, \ell_2, \ell_3, \ell_4, \ell_6, \ell_{16}), \ldots\}$.
  - $\circ$ $\deg(\sigma) = 4$ and $\deg(\omega) = \begin{cases} 4 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \cdots + \ell_{i_6} \neq 0 \\ 2 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \cdots + \ell_{i_6} = 0 \end{cases}$
  - $\circ$ $\deg(b) = \begin{cases} 2 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \cdots + \ell_{i_6} \neq 0 \\ 1 & \text{if } \ell_{i_1} + \ell_{i_2} + \ell_{i_3} + \cdots + \ell_{i_6} = 0 \end{cases}$
- $\ldots$
- *Last step:* The attacker has to solve the following system of quadratic equations in order to find the secret permutation:

$$\begin{cases} \ell_{15} = \Pi(0)\ ; & 1^{st}\ step \\ \ell_1 + \ell_4 + \ell_{16} = \ell_3 + \ell_{14} + \ell_{16} = \cdots = 0 & 2^{nd}\ step \\ \ell_1 + \ell_2 + \ell_{10} + \ell_{16} = \ell_2 + \ell_3 + \ell_{13} + \ell_{16} = \cdots = 0 & 3^{rd}\ step \\ \ell_1 + \ell_2 + \ell_3 + \ell_{12} + \ell_{16} = \ell_3 + \ell_4 + \ell_8 + \ell_{12} + \ell_{16} = \cdots = 0 & 4^{th}\ step \\ \ell_1 + \ell_2 + \ell_3 + \ell_4 + \ell_6 + \ell_{16} = \cdots = 0 & 5^{th}\ step \\ \ldots \end{cases}$$

Solving the system will allow to fully determine the secret permutation

$$\Pi(\mathcal{L}) = \mathcal{L}' = \{\alpha, \alpha^2, \alpha^3, \alpha^4, \ldots, 0, 1\}.$$

2 Toy example for the ELP evaluation attack

Consider $\mathbb{F}_{2^3}[x] = \frac{\mathbb{F}_2[x]}{x^3+x+1}$. $\mathcal{G}$ and the support $\mathcal{L} = \{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$ are public, $\mathsf{m} \in \mathbb{F}_2^k$ and $\mathcal{O}$ is the decoding oracle. We notice that if $\mathcal{L}$ is public one can find $G(x)$ such that $\mathcal{L} = \frac{\mathbb{F}_2[x]}{G(x)}$. The other way is equaly true: if $G(x)$ is public then one can easily discover $\mathcal{L}$. Suppose that the secret permutation is:

$$\Pi(\mathcal{L}) = \mathcal{L}' = \{\alpha, \alpha^3, 1, \alpha^4, \alpha^5, 0, \alpha^2, \alpha^6\} = \{\ell_i \mid i \in \{1, \ldots, 8\}\}$$

- 1^{st} *step:*
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 2$ (the positions (ℓ_j, ℓ_k) are the two non-zero positions of e).

  - ⋆ The faster step ❼ reveals the position of $\Pi(0)$: ℓ_6.
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 2$ (the positions (ℓ_6, ℓ_k) are the two non-zero positions of e).
  - ⋆ The faster step ❼ reveals the position of $\Pi(1)$: ℓ_3.

- 2^{nd} *step:*
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 3$ (the positions (ℓ_3, ℓ_j, ℓ_k) are the three non-zero positions of e).
  - ⋆ The faster step ❼ reveals all the couples (ℓ_j, ℓ_k) such that $\ell_3 + \ell_j + \ell_k = 0$. Here $(\ell_j, \ell_k) \in \{(\ell_1, \ell_2), (\ell_4, \ell_5), (\ell_7, \ell_8)\}$.

- 3^{rd} *step:*
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 3$ (the positions (ℓ_6, ℓ_j, ℓ_k) are the three non-zero positions of e).
  - ⋆ The faster step ❼ reveals all the couples (ℓ_j, ℓ_k) such that $\ell_j \ell_k = 1$. Here $(\ell_j, \ell_k) \in \{(\ell_1, \ell_8), (\ell_2, \ell_4), (\ell_5, \ell_7)\}$.

- 4^{th} *step:*
  - The attacker asks $\mathcal{O}$ to decode all the $\mathsf{z} = \mathsf{m}\mathcal{G} \oplus e$ with $\mathsf{wt}(e) = 3$ (the positions (ℓ_i, ℓ_j, ℓ_k) are the three non-zero positions of e).
  - ⋆ The faster step ❼ reveals all the triplets (ℓ_i, ℓ_j, ℓ_k) such that $\ell_i + \ell_j + \ell_k = 0$. Here $(\ell_i, \ell_j, \ell_k) \in \{(\ell_1, \ell_4, \ell_7), (\ell_1, \ell_5, \ell_8), (\ell_2, \ell_4, \ell_8), (\ell_2, \ell_5, \ell_7)\}$.
  - The attacker has to solve the following system of quadratic equations in order to find the secret permutation:

$$\begin{cases} \ell_6 = \Pi(0)\,; \ell_3 = \Pi(1) & 1^{st}\ step \\ \ell_1 + \ell_2 = \ell_4 + \ell_5 = \ell_7 + \ell_8 = 1 & 2^{nd}\ step \\ \ell_1\ell_8 = \ell_2\ell_4 = \ell_5\ell_7 = 1 & 3^{rd}\ step \\ \ell_1 + \ell_4 + \ell_7 = \ell_1 + \ell_5 + \ell_8 = 0 & 4^{th}\ step \\ \ell_2 + \ell_4 + \ell_8 = \ell_2 + \ell_5 + \ell_7 = 0 & 4^{th}\ step \end{cases}$$

Solving the system will allow to fully determine the secret permutation $\Pi(\mathcal{L}) = \{\alpha, \alpha^3, 1, \alpha^4, \alpha^5, 0, \alpha^2, \alpha^6\}$.