Timing Attacks against the Syndrome Inversion in Code-Based Cryptosystems*

Falko Strenzke

Cryptography and Computeralgebra, Department of Computer Science,
Technische Universität Darmstadt, Germany
fstrenzke@crypto-source.de

Abstract. In this work we present the first practical key-aimed timing attack against code-based cryptosystems. It arises from vulnerabilities that are present in the inversion of the error syndrome through the Extended Euclidean Algorithm that is part of the decryption operation of these schemes. Three types of timing vulnerabilities are combined to a successful attack. Each is used to gain information about the secret support, which is part of code-based decryption keys: The first allows recovery of the zero-element, the second is a refinement of a previously described vulnerability yielding linear equations, and the third enables to retrieve cubic equations.

Keywords: side channel attack, timing attack, post quantum cryptography, code-based cryptography.

1 Introduction

The McEliece PKC [1] and Niederreiter [2] Cryptosystems, built on error correcting codes, are considered immune to quantum computer attacks [3], and thus are of interest as candidates for future cryptosystems in high security applications. Accordingly, they have received growing interest from researchers in the past years and been analyzed with respect to efficiency on various platforms [4–8]. Furthermore, a growing number of works has investigated the side channel security of code-based cryptosystems [9–14].

Side channel security is a very important implementation aspect of any cryptographic algorithm. A side channel is given when a physical observable quantity that is measured during the operation of a cryptographic device, allows an attacker to gain information about a secret that is involved in the cryptographic operation. The usual observables used in this respect are the duration of the operation (timing attacks [15]), or the power consumption as a function over the time (power analysis attacks[16]).

So far, timing attacks against the decryption operation of the McEliece PKC targeting the plaintext have been developed [10, 12, 14]. In [11], a timing attack

* To the most part, this work was done in the author's private capacity, a part of the work was done at Cryptography and Computeralgebra, Department of Computer Science, Technische Universität Darmstadt, Germany

P. Gaborit (Ed.): PQCrypto 2013, LNCS 7932, pp. 217–230, 2013.

is proposed that targets the secret support that is part of the private key in code-based cryptosystems. From the time taken by the solving of the key equation the attacker learns linear equations about the support in this attack. But that work suffers from two major limitations: Neither is the information that is gained in itself sufficient for a practical attack, nor was the attack actually implemented.

This work extends on the analysis given in [11] in multiple ways: first of all, we find that a control flow ambiguity causing leakage in terms of the linear equations is manifest already in the syndrome inversion preceding the solving of the key equation in the decryption operation, and consequently the countermeasure proposed in that work is insufficient. We also show that there exists a timing side channel vulnerability in the syndrome inversion that allows the attacker to gain knowledge of the zero-element of the secret support. As an extension resp. generalization of the attack yielding linear equations, we derive a practical timing attack that lets the attacker gain cubic equations.

We then describe how to efficiently use these three vulnerabilities to build a practical attack that recovers the private key entirely. Lastly, we give results for practical executions of the timing attack on a personal computer.

2 Preliminaries

In this work, we give a brief description of the McEliece PKC, and stress those features of the decryption algorithm, that are necessary to understand the timing attack presented in this paper. A more detailed description and security considerations can be found e.g. in [17].

Goppa Codes. Goppa codes [18] are a class of linear error correcting codes. The McEliece PKC makes use of irreducible binary Goppa codes, so we will restrict ourselves to this subclass and to code lengths that are powers of two.

Definition 1. *Let the polynomial* $g(Y) = \sum_{i=0}^{t} g_i Y^i \in \mathbb{F}_{2^m}[Y]$ *be monic and irreducible over* $\mathbb{F}_{2^m}[Y]$*, and let* m*,* t *be positive integers. Then* $g(Y)$ *is called a* Goppa polynomial *(for an irreducible binary Goppa code).*

Then an irreducible binary Goppa code is defined as $\mathcal{C}(g(Y)) = \{\boldsymbol{c} \in \mathbb{F}_2^n | S_{\boldsymbol{c}}(Y) := \sum_{i=0}^{n-1} \frac{c_i}{Y-\alpha_i} = 0 \bmod g(Y)\}$*, where* $n = 2^m$*,* $S_{\boldsymbol{c}}(Y)$ *is the syndrome of* $\boldsymbol{c}$*,* $\Gamma = (\alpha_i | i = 0, \ldots, n-1)$*, the* support *of the code, where the* α_i *are pairwise distinct elements of* $\mathbb{F}_{2^m}$*, and* c_i *are the entries of the vector* $\boldsymbol{c}$*.*

The code defined in such way has length n, dimension $k \geq n - mt$ – however we restrict us to $k = n - mt$ in this work – and can correct up to t errors.

As for any linear error correcting code, for a Goppa code there exists a generator matrix $G \in \mathbb{F}_2^{k \times n}$ and a parity check matrix $H \in \mathbb{F}_2^{mt \times n}$ [19]. Given these matrices, a message $\boldsymbol{m} \in \mathbb{F}_2^k$ can be encoded into a codeword $\boldsymbol{c}$ of the code by computing $\boldsymbol{c} = \boldsymbol{m}G$, and the syndrome $\boldsymbol{s} \in \mathbb{F}_2^{mt}$ of a (potentially distorted) codeword can be computed as $\boldsymbol{s} = \boldsymbol{c}H^T$. Here, we do not give the formulas for the computation of these matrices as they are of no importance for the understanding of the attack developed in this work. The interested reader, however, is referred to [19].

Overview of the McEliece PKC. In this section we give a brief overview of the McEliece PKC. The McEliece *secret key* consists of the Goppa polynomial $g(Y)$ of degree t and the support $\Gamma = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$, i.e. a permutation of $\mathbb{F}_{2^m}$, together they define the secret code $\mathcal{C}$. The *public key* is given by the public $n \times k$ generator matrix $G_p = SG$ over $\mathbb{F}_2$, where G is a generator matrix of the secret code $\mathcal{C}$ and S is a non-singular $k \times k$ matrix over $\mathbb{F}_2$, the purpose of which is to bring G_p into reduced row echelon form, i.e. $G_p = [\mathbb{I}|G_2]$, which results in a more compact public key [4].

Note that in the original definition of the McEliece PKC [1], the support is chosen to be in lexicographical ordering, but instead the public key is chosen as $G_p = SGP$, where P is a random permuation matrix. These two descriptions are completely equivalent: P corresponds to the permutation that has to be applied to a lexicographical ordered support Γ to produce the randomized secret support as it is defined above. Accordingly, the attack described in this work that attacks the secret support can alternatively be seen as an attack against the secret permutation.

The *encryption* operation allows messages $\boldsymbol{m} \in \mathbb{F}_2^k$. A random vector $\boldsymbol{e} \in \mathbb{F}_2^n$ with hamming weight $\mathrm{wt}(\boldsymbol{e}) = t$ has to be created. Then the ciphertext is computed as $\boldsymbol{z} = \boldsymbol{m}G_p + \boldsymbol{e}$.

The Decryption is given in Algorithm 1. It makes use of the error correction algorithm, given by the Patterson Algorithm [20], shown in Algorithm 2. In Step 1 of this algorithm, the syndrome vector is computed by multiplying the ciphertext by the parity check matrix, and then turned into the syndrome polynomial $S(Y)$ by interpreting it as an $\mathbb{F}_{2^m}^t$ element and multiplying it with the vector of powers of Y. The Patterson Algorithm furthermore uses an algorithm for finding roots in polynomials over $\mathbb{F}_{2^m}$ (root_find()), and the Extended Euclidean Algorithm (EEA) for polynomials with a break condition based on the degree of the remainder, given in Algorithm 3. The root finding can e.g. be implemented as an exhaustive search on $\mathbb{F}_{2^m}$. Please note that all polynomials appearing in the algorithms have coefficients in $\mathbb{F}_{2^m}$.

The Niederreiter PKC [2] is a cryptosystem that is slightly different from the McEliece PKC, however there also an error vector is chosen during the encryption and decryption features the syndrome decoding. Since these features are, as we shall see, the preconditions for our attack, it is equally applicable to the Niederreiter PKC.

In the following, we turn to those details, that are relevant for the side channel issues we are going to address in Section 3. Please note that the error locator polynomial $\sigma(Y)$, which is determined in Step 4 of Algorithm 2, has the following form:

$$\sigma(Y) = \prod_{j \in \mathcal{E}} (Y - \alpha_j) = \sum_{i=0}^{t} \sigma_i Y^i. \tag{1}$$

where $\mathcal{E}$ is the set of those indexes i, for which $e_i = 1$, i.e. those elements of $\mathbb{F}_{2^m}$ that correspond to the error positions in the error vector. The determination of the error vector in Step 6 of Algorithm 2 makes use of this property. Accordingly, $\deg(\sigma(Y)) = \mathrm{wt}(\boldsymbol{e})$ if $\mathrm{wt}(\boldsymbol{e}) \leq t$ holds.

Algorithm 1. The McEliece Decryption Operation

Require: the McEliece ciphertext $\boldsymbol{z} \in \mathbb{F}_2^n$
Ensure: the message $\boldsymbol{m} \in \mathbb{F}_2^k$
1: $\boldsymbol{e} \leftarrow \text{err\_corr}(\boldsymbol{z}, g(Y))$
2: $\boldsymbol{m}' \leftarrow \boldsymbol{z} + \boldsymbol{e}$
3: $\boldsymbol{m} \leftarrow$the first k bits of $\boldsymbol{m}'$
4: return $\boldsymbol{m}$

Algorithm 2. The McEliece error correction with the Patterson Algorithm (err_corr$(\boldsymbol{z}, g(Y))$)

Require: the distorted code word $\boldsymbol{z} \in \mathbb{F}_2^n$, the secret Goppa polynomial $g(Y)$ and secret support $\Gamma = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$
Ensure: the error vector $\boldsymbol{e} \in \mathbb{F}_2^n$
1: $S(Y) \leftarrow \boldsymbol{z} H^\top \left(Y^{t-1}, \cdots, Y, 1\right)^\top$
2: $\tau(Y) \leftarrow \sqrt{S^{-1}(Y) + Y} \bmod g(Y)$
3: $(a(Y), b(Y)) \leftarrow \text{EEA}\left(\tau(Y), g(Y), \lfloor \frac{t}{2} \rfloor\right)$
4: $\sigma(Y) \leftarrow a^2(Y) + Y b^2(Y)$
5: $\mathcal{E} = \{E_0, \ldots, E_{t-1}\} \leftarrow \text{rootfind}(\sigma(Y))$ // if α_i is a root, then $\mathcal{E}$ contains i
6: $\boldsymbol{e} \leftarrow \boldsymbol{v} \in \mathbb{F}_2^n$ with $v_i = 1$ if and only if $i \in \mathcal{E}$
7: **return** $\boldsymbol{e}$

3 Analysis of Timing Side Channels in the Syndrome Inversion

We now explain three different vulnerabilities present in the syndrome decoding. To this end, we first explore certain properties of the syndrome inversion by the use of the EEA during the code-based decryption operation.

3.1 Properties of the Syndrome Inversion

The syndrome polynomial is defined as

$$S(Y) \equiv \sum_{i=1}^{w} \frac{1}{Y \oplus \epsilon_i} \equiv \frac{\Omega(Y)}{\sigma(Y)} \bmod g(Y) \tag{2}$$

Here, w is the Hamming weight of the error vector $\boldsymbol{e}$ and the $\{\epsilon_i | i \in \{1, \ldots w\}\}$ denote the support elements associated with the indexes of those bits in the error vector having value one in arbitrary ordering, i.e., for instance, if the bits found at the index j and k in the error vector have value one, then $\epsilon_1 = \alpha_j$, $\epsilon_2 = \alpha_k$ and so on. The identification of the error locator polynomial $\sigma(Y)$ in the denominator is simply a result of the form of the common denominator of all sum terms. In the McEliece PKC Decryption, during the error correction, Alg. 2, Step 2, $S^{-1}(Y)$ is computed by invoking Alg. 3 as EEA$(g(Y), S(Y), 0)$. But it is known that in case of $w \leq t/2$ instead it is possible to find $\sigma(Y)$ already at this

Algorithm 3. The Extended Euclidean Algorithm (EEA($r_{-1}(Y)$, $r_0(Y)$, d))

Require: the polynomials $r_{-1}(Y)$ and $r_0(Y)$, with $\deg(r_0(Y)) < \deg(r_{-1}(Y))$
Ensure: two polynomials $r_M(Y)$, $b_M(Y)$ satisfying $r_M(Y) = b_M(Y)r_0(Y) \bmod r_{-1}(Y)$ and $\deg(r_0(Y)) \leq \lfloor \deg(r_{-1})/2 \rfloor$

1: $b_{-1} \leftarrow 0$
2: $b_0 \leftarrow 1$
3: $i \leftarrow 0$
4: **while** $\deg(r_i(Y)) > d$ **do**
5: $\quad i \leftarrow i + 1$
6: $\quad (q_i(Y), r_i(Y)) \leftarrow r_{i-2}(Y)/r_{i-1}(Y)$ // polynomial division with quotient q_i and remainder r_i
7: $\quad b_i(Y) \leftarrow b_{i-2}(Y) + q_i(Y)b_{i-1}(Y)$
8: **end while**
9: $M \leftarrow i$
10: **return** $(r_M(Y), b_M(Y))$

stage by invoking Alg. 3 as EEA$(g(Y), S(Y), \lfloor t/2 \rfloor - 1)$, i.e. with $r_{-1}(Y) = g(Y)$ and $r_0(Y) = S(Y)$ and breaking once $\deg(r_i(Y)) \leq (t/2) - 1$. Then, it returns $\delta\sigma(Y) = b_M(Y)$ and furthermore $\delta\Omega(Y) = r_M(Y)$, where $\delta \in \mathbb{F}_{2^m}$ and M is the number of iterations performed by the EEA [21].

Given this form of the $S(Y)$, we can make a statement about the maximally possible number of iterations in the EEA used to compute $S^{-1}(Y) \equiv \sigma(Y)/\Omega(Y) \bmod g(Y)$. As already mentioned, the actual invocation of the syndrome inversion is EEA$(g(Y), S(Y), 0)$. But the above explained fact that we could stop at $\deg(r_i(Y)) \leq (t/2) - 1$ means that there is one iteration in the EEA where $r_i(Y) = \delta\Omega(Y)$ and $b_i(Y) = \delta\sigma(Y)$, in case of $w \leq (t/2) - 1$.

Theorem 1. *Assume a Goppa Code defined by $g(Y)$ and Γ. When Alg. 3 is invoked as* EEA$(g(Y), S(Y), 1)$ *with* $S(Y) \equiv \frac{\Omega(Y)}{\sigma(Y)} \bmod g(Y)$*, and the error vector $\boldsymbol{e}$ corresponding to $S(Y)$ satisfies* $\mathrm{wt}(\boldsymbol{e}) \leq (\deg(g(Y))/2) - 1$*, then for the number of iterations in Alg. 3 we find:*

$$M \leq M_{\max} = \deg(\Omega(Y)) + \deg(\sigma(Y))$$

Proof. Regard the iteration where $r_j(Y) = \delta\Omega(Y)$ and $b_j(Y) = \delta\sigma(Y)$. Since according to Alg. 3 the degree of $b_j(Y)$, starting from zero, increases at least by one in each iteration, we find $j \leq \deg(\sigma(Y))$. From here on, the degree of $r_j(Y) = \delta\Omega(Y)$ is decreased by at least one in each subsequent iteration down to $\deg(r_M(Y)) = 0$, i.e. $M - j \leq \deg(\Omega(Y))$, giving $M = M - j + j \leq \deg(\Omega(Y)) + \deg(\sigma(Y))$.

Because in the following we are only interested in the derivation of equations of the form $\sigma_i = 0$ for a specific value of i, we will ignore the constant δ from here on.

3.2 Linear Equations from $w = 4$ Error Vectors

We now investigate the effect of the above results for the case where ciphertexts created with error vectors of Hamming weight four are input to the decryption operation.

In the case of $w = 4$ the syndrome polynomial is of the form:

$$S(Y) \equiv \frac{\Omega(Y)}{\sigma(Y)} \equiv \sum_{i=1}^{4} \frac{1}{Y \oplus \epsilon_i} \equiv \frac{\sigma_3 Y^2 \oplus \sigma_1}{Y^4 \oplus \sigma_3 Y^3 \oplus \sigma_2 Y^2 \oplus \sigma_1 Y \oplus \sigma_0} \bmod g(Y), \quad (3)$$

where $\epsilon_i \in \mathbb{F}_{2^m}, i \in 1, \ldots, 4$ denote the four elements of the support associated with the error positions. Furthermore, in the right hand side of Eq. (3), which is found by bringing all four sum terms to their common denominator, we have

$$\sigma_3 = \epsilon_1 \oplus \epsilon_2 \oplus \epsilon_3 \oplus \epsilon_4.$$

With the aim of finding a timing vulnerability revealing certain coefficients of $\sigma(Y)$ and thus information about the secret support, we now analyze the connection between the number of iterations and their complexity on the one hand and the degree of $\Omega(Y)$ on the other. Regarding $\Omega(Y)$ for the case $w = 4$ we find that the coefficient to the highest power of Y is given by $\sigma_3 = \epsilon_1 \oplus \epsilon_2 \oplus \epsilon_3 \oplus \epsilon_4$. If $\sigma_3 = 0$, then the degree of $\Omega(Y)$ is zero, otherwise it is two. This means that in the case of $\sigma_3 = 0$ the maximal number of iterations in the inversion is four, in contrast to six in the general case. Table 1 gives an overview of the individual iterations in the syndrome inversion EEA when $w = 4$, where it is assumed that for each iteration $\deg(q_i(Y)) = 1$, i.e. the case where the maximal number of iterations $M_{\max}$is executed. In the majority of the cases $M_{\max}$ iterations occur, i.e. six when $\deg(\Omega(Y)) = 2$ and four when $\deg(\Omega(Y)) = 0$. But with probability about $1/n$ in each iteration a larger degree of the quotient polynomial $q_i(Y)$ occurs, accordingly then $M < M_{\max}$. With the aim of assessing the reliability of the differences in running time allowing to identify the case $\deg(\Omega(Y)) = 0$, we examine whether $M < M_{\max}$ might lead to timings for $\deg(\Omega(Y)) = 2$ as low as for $\deg(\Omega(Y)) = 0$. We immediately find that the fifth iteration, which is only executed in the case $\sigma_3 = 0$, features a much more complex multiplication $q_5(Y)b_4(Y)$ than all the other iterations.

The control flow for the second EEA invocation, i.e. the solving of the key equation, for the case $w = 4$ has been analyzed in [11], there it is shown that in the case of $\sigma_3 = 0$ the number of iterations N is zero, whereas in the case $\sigma_3 \neq 0$ it is one. In that work, a countermeasure is proposed that removes the possibility to exploit the according timing differences in the second EEA invocation. However, due the fact that, as shown above, timing differences reveal $\sigma_3 = 0$ already in the syndrome inversion EEA, the countermeasure proposed in [11] is insufficient.

Experimental results confirm that taken together, the timing differences emerging in both EEA applications, i.e. the syndrome inversion and the key equation solving, actually allow for reliable distinction of $\deg(\Omega(Y))$ being zero or non-zero, and thus the attacker is able to learn linear equations of the form $\sigma_3 = \sum_{i=1}^{4} \epsilon_i = 0$. Remember that through the choice of the error vector during

encryption, he chooses the indexes j_i with $i = 1, \ldots, 4$ of the support elements $\alpha_{j_i} = \epsilon_i$ according to the definition of the ϵ_i notation for the support elements.

Table 1. Overview of the iterations in the syndrome inversion EEA for Hamming weight four error vectors. If $\deg(\Omega(Y)) = 2$, $M_{\max}$, the maximal number of iterations is six, otherwise, if $\deg(\Omega(Y)) = 0$, we have $M_{\max} = 4$.

i	$\deg(q_i(Y))$	$\deg(b_i(Y))$	$\deg(r_i(Y))$
1	1	1	t-1
2	1	2	t-2
3	1	3	t-3
4	1	**4**	**2 (or 0)**
5	t - 5	t - 1	1
6	1	t	0

3.3 Cubic Equations from $w = 6$ Error Vectors

The vulnerability found for $w = 4$ error vectors can be generalized to any even value of w. For the attack that is subject of this work, we also employ the case $w = 6$. There, we find that the syndrome polynomial according to Eq. (2) is of the form

$$S(Y) \equiv \frac{\Omega(Y)}{\sigma(Y)} \equiv \frac{\sigma_5 Y^4 \oplus \sigma_3 Y^2 \oplus \sigma_1}{Y^6 \oplus \sigma_5 Y^5 \oplus \sigma_4 Y^4 \oplus \sigma_3 Y^3 \oplus \sigma_2 Y^2 \oplus \sigma_1 Y + \sigma_0} \bmod g(Y), \tag{4}$$

where

$$\sigma_3 = \sum_{j=3}^{6} \sum_{k=1}^{j-1} \sum_{l=1}^{k-1} \epsilon_j \epsilon_k \epsilon_l, \tag{5}$$

$$\sigma_5 = \sum_{i=1}^{6} \epsilon_i. \tag{6}$$

As in case of $w = 4$, $\deg(\Omega(Y)) = 0$ implies zero iterations in the key equation EEA. Furthermore, it is again the most complex iteration of the syndrome inversion EEA that is skipped if $\deg(\Omega(Y)) = 0$. The difference to $w = 4$ is that here two coefficients of $\sigma(Y)$, i.e σ_3 and σ_5, have to be zero for this to happen.

Thus from detecting $\deg(\Omega(Y)) = 0$ the attacker can learn the equations $\sigma_3 = 0$ and $\sigma_5 = 0$. However, since from the vulnerability presented in Sec. 3.2 it is already possible for the attacker to learn linear equations about the secret support, the value of the "$w = 6$" vulnerability lies in the equation $\sigma_3 = 0$, which can be learned through a timing side channel analogously to the case "$w = 4$".

3.4 The Zero Element of the Support from $w = 1$ Error Vectors

For $w = 1$ the whole control flow in Patterson's Algorithm is very simple and unambiguous on a high level: $S(Y) \equiv \frac{1}{Y \oplus \epsilon_1} \bmod g(Y)$, $S^{-1}(Y) = Y \oplus \epsilon_1$,

Algorithm 4. Polynomial Division poly_div$(n(Y), d(Y))$

Require: the polynomials $n(Y), d(Y)$ with $\deg(n(Y)) \geq \deg(d(Y))$
Ensure: two polynomials $s(Y)$, $q(Y)$ with $q(Y)d(Y) + s(Y) = n(Y)$ and $\deg(s(Y)) < \deg(d(Y))$

```
s_{-1}(Y) ← n(Y)
s_0(Y) ← d(Y)
q_0(Y) ← 0
i ← 0
while deg(s_i(Y)) ≥ deg(d(Y)) do
    i ← i + 1
    a_i ← s_{i-2,deg(s_{i-2}(Y))} / s_{i-1,deg(s_{i-1}(Y))}
    f_i ← deg(s_{i-2}(Y)) − deg(s_{i-1}(Y))
    q_i(Y) = q_{i-1} + a_i Y^{f_i}
    s_i ← s_{i-2}(Y) − a_i s_{r-1}(Y) Y^{f_i}
end while
return (q_i(Y), s_i(Y))
```

$\tau(Y) = \sqrt{\epsilon_1}$, $a(Y) = \tau(Y)$, $b(Y) = 1$, $\sigma(Y) = Y \oplus \epsilon_1$. The polynomial inversion is, according to Theorem 1, performed in exactly one iteration. But there is an ambiguous control flow within the polynomial division given in Alg. 4, that is executed within this EEA iteration: We find $q_1(Y) = Y$ because there is no alternative to $\deg(S(Y)) = t - 1$. In Alg. 4, $s_{i,j}$ denotes the coefficient to Y^j in $s_i(Y)$. If $\epsilon_1 = 0$, then the division has to stop at this point. Otherwise, a second iteration is performed giving $q_2(Y) = Y \oplus \epsilon_1$. Thus, if the timing difference resulting from the different number of iterations in the division is detectable, the index of z of the secret support element $\alpha_z = 0$ can be found.

4 Combining the "$w = 1$", "$w = 4$", and "$w = 6$" Vulnerabilities to a practical Attack

In this section we explain the construction of a practical attack based on the vulnerabilities shown in Sections 3.2, 3.3 and 3.4.

4.1 Description of the Attack Procedure

Step 1. By performing the respective queries on the decryption device with "$w = 4$" error vectors, a rank $n - m - 1$ linear equation system is build. The experimental results from [11] already showed that this is the maximal rank that can be achieved from the linear equations. Afterwards, the index of the zero element, α_z is determined through the "$w = 1$" vulnerability. In the majority of the cases, this information increases the rank of the equation system to $n - m$. In the rare cases when the rank remains at $n - m - 1$, the attack's on-line and off-line complexity is increased by a factor of n.

In the following, we assume that we have an equation system of rank $n - m$. This is the highest possible rank for a homogeneous linear equation system

describing a permutation of $\mathbb{F}_{2^m}$, since there must be m linearly independent basis elements. Accordingly, by bringing the linear equation system into reduced row echelon form, we find the elements associated with the m rightmost columns must be a basis $\{\beta_i\}$:

$$\begin{array}{cccccccc|ccc}
\alpha_0 & \alpha_1 & \dots & \alpha_i & \dots & \alpha_{n-m-3} & \alpha_{n-m-2} & \beta_0 & \dots & \beta_{m-1} \\
\hline
1 & 0 & \dots & 0 & \dots & 0 & 0 & X & \dots & X \\
\vdots & & & & & & & & & \\
0 & 0 & \dots & 1 & \dots & 0 & 0 & X & \dots & X \\
\vdots & & & & & & & & & \\
0 & 0 & \dots & 0 & \dots & 0 & 1 & X & \dots & X
\end{array}$$

Step 2. At this point for each element α_i we know the corresponding B_i with $\alpha_i = \sum_{j \in B_j} \beta_i$, i.e. its representation in the chosen basis. If the values of all basis elements β_i were known, then the values of all α_i would be set as well and the support was recovered. Accordingly, the next step in the attack is to collect cubic equations according to Eq. (5) in a way that allows for efficient guessing resp. solving for the values of the β_i. To this end, the first set C_1 of "$w = 6$" equations is created by the employment of error vectors involving error positions corresponding to ϵ_i, $i = 1, \dots, 6$, where the following conditions hold:

1. $\epsilon_i \in \mathrm{span}(\{\beta_{\mathrm{s}_1}, \beta_{\mathrm{g}_1}, \beta_{\mathrm{g}_2}, \beta_{\mathrm{g}_3}\})$. These are four arbitrarily chosen basis elements, where β_{s_1} denotes the one to be solved for in the resulting equation according to Eq. (5). The reason for this initial set of basis elements having cardinality four is that this is the lowest cardinality allowing to satisfy all the conditions in the following items.
2. $\sum_{i=1}^{6} \epsilon_i = 0$. This qualifies the error vector for the possibility of $\deg(\Omega(Y)) = N = 0$ according to Eq. (6) in the sense that $\sigma_5 = 0$ is already ensured. As a result, in contrast to the case of random $w = 6$ error vectors that have a probability for $\deg(\Omega(Y)) = N = 0$ in the domain of $1/n^2$, for these candidates this probability is about $1/n$.
3. Exactly two of the ϵ_i contain β_{s_1}. The reason for this constraint is to keep the process of solving, the details of which we shall see shortly, as simple as possible. Specifically, the twofold occurrence of β_{s_i} leads to a quadratic equation for β_{s_i}.

Candidate error vectors $\boldsymbol{e}$ that meet these conditions are used to build ciphertexts which are input to the decryption device; and from the timing of the decryption, it is inferred whether actually $\deg(\Omega(Y)) = N = 0$ occurred. The number of such equations to be collected for one β_{s_i} is given by c_i, which is a parameter for the attack.

After c_1 equations are found for β_{s_1}, the second set of equations is build in the same way as the first, the only differences being that the basis element to be solved for now is $\beta_{s_2} \notin \{\beta_{s_1}, \beta_{g_1}, \beta_{g_2}, \beta_{g_3}\}$, and first condition becomes $\epsilon_i \in \mathrm{span}(\{\beta_{\mathrm{s}_1}, \beta_{\mathrm{g}_1}, \beta_{\mathrm{g}_2}, \beta_{\mathrm{g}_3}, \beta_{\mathrm{s}_2}\})$. In this manner successively sets of cubic equations for $m-3$ different β_{s_i} are collected until the equations in the last set involve all β_i.

Step 3. In this step the solving resp. guessing is performed. Let those two ϵ_i that contain β_{s_i} according to the third condition in Step 4 always be ϵ_1 and ϵ_2. From the conditions given in Step 4, and Eq. (5) we have for each β_{s_i} a quadratic equation

$$a\beta_{s_i}^2 + b\beta_{s_i} + c = 0, \tag{7}$$

with $a = \sum_{j=3}^{6} \epsilon_i$, $b = (\epsilon_1+\epsilon_2)a$, $c = (\epsilon_1-\beta_{s_i})(\epsilon_2-\beta_{s_i})a+(\epsilon_1+\epsilon_2)\sum_{j=4}^{6}\sum_{k=3}^{j-1}\epsilon_j\epsilon_k$ $+\sum_{j=5}^{6}\sum_{k=4}^{j-1}\sum_{l=3}^{k-1}\epsilon_j\epsilon_k\epsilon_l$ (for clarity, in these formulas we provide "+" and "−" even though both amount to "⊕"). Such a quadratic equation has two solutions for β_{s_i}.

The solving is performed as follows: enumerate the initial guesses, i.e. all the possible combinations of the values for $\beta_{g_1}, \beta_{g_2}, \beta_{g_3}$. Here, and for the subsequent guesses, since we are looking for linearly independent $\mathbb{F}_{2^m}$ elements, it holds that

$$\beta_{g_i} \notin \mathrm{span}(\{\beta_{\mathrm{g_1}}, \ldots, \beta_{\mathrm{g_{i-1}}}\}), \tag{8}$$

where we imply the convention $\beta_{s_i} = \beta_{g_{i+3}}$.

For each such combination of values for $\beta_{g_1}, \beta_{g_2}, \beta_{g_3}$ the roots of each equation in C_1 are potential candidates for the value of β_{s_1}. However, additionally to the restriction from Eq. (8), those roots that are found only for a subset of C_1 are discarded. This is wherein the value of a choice $c_i = |C_i| > 1$ lies. The larger the c_i are chosen, the higher is the on-line effort of the attack (more cubic equations have to be collected), but the off-line effort is reduced as the number possible solutions for each β_{s_i} is decreased.

The remaining roots are iterated over to find the possible solutions for β_{s_2} by solving the equations in C_2, which in turn are used to compute the possible values of β_{s_3}, etc. Whenever in such a chain of guesses a solution for all β_{s_i} is found, a guess for the whole support $\Gamma = (\alpha_i | \alpha_i = \sum_{j\in B_i} \beta_j)$ is implied, which has to be checked by a means of key recovery, as described in [13].

4.2 Experimental Results

We conducted the attack with the following measurement setup on an Intel Core 2 Duo x86 platform: from the attack program, the decryption function was called with the attack ciphertexts as input, and the decryption time was measured with the CPU's cycle counter.

Because the cycle counts measured for a deterministic operation of the duration of a code-based decryption vary considerably on such CPUs, a specific strategy has to be used to identify positives, by which we refer to $\epsilon_1 = 0$ for $w = 1$ and $\deg(\Omega(Y)) = 0$ for $w = 4$ and $w = 6$, i.e. those cases that yield an equation for the attacker. Specifically, an approximate model for cycle counts on modern x86 CPUs like the Core2 Duo is a hypothetical constant cycle count associated with the operation which is increased by a random delay on every execution of an operation. Because in all three different attack types the positives, from the algorithmic point of view, are executed faster than the negatives, the following classification strategy can be used: Prior to the attack a training

phase is carried out where the minimal cycle counts for positives are determined as well as the minimal cycle counts for negatives (using a different secret key than during the attack). Then the border below which an operation is classified as a positive during the attack is set as the mean of these two values. We refer to the distance between the minimal cycle counts for positives and the minimal cycle counts for negatives as the cycles gap. Clearly, a larger such gap increases the probability for finding positives. Furthermore, the above approximate model for the cycle counts on the employed CPU is lacking other effects that could be observed in our experiments: during the execution of the attack the previously determined maximal and minimal cycle counts for the two classes of operations seem to be subject to an "upwards drift", i.e. they tend to successively increase over time but sometimes also drop again approximately to the initial levels after some time.

Table 2 summarizes the results for single attack runs with different code parameters. The rows labeled "cycles gap ..." indicate the above discussed gaps. We found that gaps of a couple of hundreds cycles that are characteristic for the $w = 1$ vulnerability tend to cause problems in the detection of positives, i.e. in some runs due to the mentioned drift of the cycle counts the zero support element could not be determined, while the considerably larger gaps for the $w = 4$ and $w = 6$ vulnerabilities allow for reliable detection of positives.

The rows labeled "number of queries ..." show the number of decryption operations that had to be executed with ciphertexts created with error vectors of the respective weight in the course of a single run of the attack.

"number of final verifications" is the number of the guesses for the complete support that are output by the attack. We did not implement an actual verification, but simply compare the guess for the support with the correct support Γ. As already mentioned, in [13] the procedure that had to be used in a real life attack is described. It involves only some linear algebra operations on the public key and the invocation of an EEA and would not perceptibly increase the time for solving, given the small numbers of such final verifications occurring in the attacks.

The time for the solving step is given in the last row. From the theory, one expects an increase of the solving time by a factor of about eight for each increase of m by one. The reason is that the number of initial guesses, i.e. the number of combinations of values that can be chosen for β_{g_1}, β_{g_2} and β_{g_3} is roughly n^3, and all $\mathbb{F}_{2^m}$ operations, including the solving of the quadratic equations [22] Eq. (7), are done with the help of lookup tables, and thus execute in constant time.

The number of equations gathered per β_{s_i} were chosen as $C_1 = 1$, $C_2 = 2$, $C_3 = 4$, i.e. chosen as the double of the previous count, up to a maximal value of 16, i.e. $C_i = 16$ for $i \geq 5$.

As previously mentioned, in the rare cases where the knowledge about the zero-element of the support does not increase the rank of the equation system, Steps 2 and 3 would have to be repeated about up to n times, for these cases stronger hardware would be needed to keep the solving time in reasonable margins.

Table 2. Experimental results for single runs of the attack. Refer to the text for explanations

	$m = 9,\ t = 33$	$m = 10,\ t = 40$
cycles gap $w = 1$	≈ 400	≈ 600
cycles gap $w = 4$	$\approx 13,000$	$\approx 19,000$
cycles gap $w = 6$	$\approx 17,000$	$\approx 23,000$
number of queries for $w = 1$ (Step 1)	3,575,494	11,782,695
number of queries for $w = 4$ (Step 1)	1,517,253	2,869,424
number of queries for $w = 6$ (Step 2)	374,927	1,837,125
number of final verifications (Step 3)	$\approx 8,000$	$\approx 2,000$
running time for solving on 1 GHz x86 CPU (Step 3)	3h	28h

5 Conclusion

The results of this work show that timing attacks based on control flow vulnerabilities in the syndrome inversion and the key equation EEA are a threat to the confidentiality of the secret key. In the chosen measurement setup, the attack has been proved to be practical. Apart from the recovery of the zero-element of the support, the cycles gaps between the controls flows that have to be distinguished are rather large, and thus remote timing attacks seem feasible too. If the zero-element remains unknown, the on-line and off-line attack complexity can still be managed with appropriate hardware.

The question of countermeasures against this attack has not been explicitly addressed in this work, but two possibilities seem to suggest themselves: the first would be similar to the countermeasures given in [12], where "premature" abortion of the key equation solving EEA is prevented by enforcing the "missing" iterations. This however is a delicate undertaking, as even the smallest timing differences have to be prohibited and thus the complexity of the individual iterations must be accounted for (consider for instance the "$w = 1$ attacks" from Section 3.4).

The second option would be stronger in this respect: there, we alter the cryptosystem's parameter specification: during the encryption, only $t - 1$ errors are added, and prior to the standard decryption operation, another "bit flip error" is applied, the position of which should be the same for repeated decryptions of a certain ciphertext but otherwise appear as random, and thus should be pseudorandomly derived from the ciphertext and a constant secret value (for instance a hash value of the secret key). This approach would guarantee a pervasive alteration of the decryption operation, however it demands an increase of security parameters to compensate for the lower error weight used during encryption.

References

1. McEliece, R.J.: A public key cryptosystem based on algebraic coding theory. DSN Progress Report 42-44, 114–116 (1978)
2. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. Problems Control Inform. Theory 15(2), 159–166 (1986)

3. Bernstein, D.J., Buchmann, J., Dahmen, E.: Post Quantum Cryptography. Springer Publishing Company, Incorporated (2008)
4. Biswas, B., Sendrier, N.: McEliece Cryptosystem Implementation: Theory and Practice. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 47–62. Springer, Heidelberg (2008)
5. Heyse, S.: Low-Reiter: Niederreiter Encryption Scheme for Embedded Microcontrollers. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 165–181. Springer, Heidelberg (2010)
6. Eisenbarth, T., Güneysu, T., Heyse, S., Paar, C.: MicroEliece: McEliece for Embedded Devices. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 49–64. Springer, Heidelberg (2009)
7. Shoufan, A., Wink, T., Molter, G., Huss, S., Strenzke, F.: A Novel Processor Architecture for McEliece Cryptosystem and FPGA Platforms. In: Proceedings of the 2009 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2009, pp. 98–105. IEEE Computer Society, Washington, DC (2009)
8. Strenzke, F.: A Smart Card Implementation of the McEliece PKC. In: Samarati, P., Tunstall, M., Posegga, J., Markantonakis, K., Sauveron, D. (eds.) WISTP 2010. LNCS, vol. 6033, pp. 47–59. Springer, Heidelberg (2010)
9. Molter, H.G., Stöttinger, M., Shoufan, A., Strenzke, F.: A Simple Power Analysis Attack on a McEliece Cryptoprocessor. Journal of Cryptographic Engineering (2011)
10. Strenzke, F., Tews, E., Molter, H., Overbeck, R., Shoufan, A.: Side Channels in the McEliece PKC. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 216–229. Springer, Heidelberg (2008)
11. Strenzke, F.: A Timing Attack against the Secret Permutation in the McEliece PKC. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 95–107. Springer, Heidelberg (2010)
12. Shoufan, A., Strenzke, F., Molter, H., Stöttinger, M.: A Timing Attack against Patterson Algorithm in the McEliece PKC. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 161–175. Springer, Heidelberg (2010)
13. Heyse, S., Moradi, A., Paar, C.: Practical power analysis attacks on software implementations of mceliece. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 108–125. Springer, Heidelberg (2010)
14. Avanzi, R., Hoerder, S., Page, D., Tunstall, M.: Side-channel attacks on the mceliece and niederreiter public-key cryptosystems. J. Cryptographic Engineering 1(4), 271–281 (2011)
15. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
16. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
17. Engelbert, D., Overbeck, R., Schmidt, A.: A Summary of McEliece-Type Cryptosystems and their Security. Journal of Mathematical Cryptology 1(2), 151–199 (2006)
18. Goppa, V.D.: A new class of linear correcting codes. Problems of Information Transmission 6, 207–212 (1970)

19. MacWilliams, F.J., Sloane, N.J.A.: The theory of error correcting codes. North Holland (1997)
20. Patterson, N.: Algebraic decoding of Goppa codes. IEEE Trans. Info. Theory 21, 203–207 (1975)
21. Sugiyama, Y., Kasahara, M., Hirasawa, S., Namekawa, T.: A method for solving key equation for decoding goppa codes. Information and Control 27(1), 87–99 (1975)
22. Biswas, B., Herbert, V.: Efficient Root Finding of Polynomials over Fields of Characteristic 2. In: WEWoRK (2009), hal.inria.fr/hal-00626997/PDF/tbz.pdf