# Side-Channel Attacks on the McEliece and Niederreiter Public-Key Cryptosystems

Roberto M. Avanzi[1], Simon Hoerder[1,2], Dan Page[2], Michael Tunstall[2]
roberto.avanzi@ruhr-uni-bochum.de
{hoerder,page,tunstall}@compsci.bristol.ac.uk

[1] HGI and Faculty of Mathematics, Ruhr-University Bochum.
[2] Department of Computer Science, University of Bristol*.

**Abstract.** Research within "post-quantum" cryptography has focused on development of schemes that resist quantum cryptanalysis. However, if such schemes are to be deployed, practical questions of efficiency and physical security should also be addressed; this is particularly important for embedded systems. To this end, we investigate issues relating to side-channel attack against the McEliece and Niederreiter public-key cryptosystems, for example improving those presented by [19], and novel countermeasures against such attack.

**Key words:** public-key cryptography, McEliece, Niederreiter, embedded systems, side-channel attack.

## 1 Introduction

The availability of a heterogeneous, i.e., structurally diverse, range of cryptosystems has significant advantages. For example, diversity in security properties that underpin said cryptosystems helps to insulate users from advances in cryptanalysis. This fact has been amplified by the advent of quantum computing. It is hard to assess when practical quantum computers will be available, but much easier to see that their development will have a profound impact on classical cryptography in the long term. Concrete examples of quantum cryptanalysis include Shor's algorithms for factoring and discrete logarithms [15] which, given a suitable quantum computer, could threaten the security of RSA and elliptic curve cryptosystems (the elliptic curve case has been investigated more closely by Proos and Zalka [13]). Within this context two "post-quantum" research areas have emerged in classical cryptography, namely

1. the design of new cryptosystems that are immune to quantum cryptanalysis and based on a more diverse range of security properties, and
2. the search for, and practicalisation of, existing cryptosystems that satisfy the same requirements.

The well-studied structure and security properties of the hash based signature schemes proposed by Merkle [9] are a good example of the second case. Likewise, the first public-key encryption scheme of this type, published in 1978 and proposed by McEliece [8], has security properties based on the intractability of decoding generic linear error correcting

* The work described in this paper has been supported in part by EPSRC grants EP/H001689/1 and EP/F039638/1.

codes. The scheme scrambles the structure of a Classical Goppa Code (CGC) pseudo-randomly so that efficient error correction is only possible with knowledge of the original code structure.

However, the same diversity can also imply disadvantages. In particular, any post-quantum cryptosystem must be practical *as well as* secure against quantum cryptanalysis. In this respect, it is imperative that cryptographic engineering keeps pace with developments in post-quantum cryptography; in an embedded context specifically, this will allow cryptosystems to be efficiently realised while also resisting physical attack. With this motivation in mind, computational efficiency of the McEliece scheme on embedded platforms (more specifically, on an 8-bit AVR processor) was studied by Eisenbarth et al. [4]. The physical security properties of the McEliece scheme is less understood. Strenzke et al. [19], [16], [20] offer examples of timing/reaction attacks based on the error weight. Heyse et al. [5] offer examples of SPA attacks.

In this paper, we extend existing research that investigates the physical security of McEliece [8] and Niederreiter [10] public-key encryption schemes by making two main contributions. Beforehand, both schemes (Section 2) are outlined and notation is introduced. The first contribution (Section 3) improves a previous timing attack of Strenzke et al. and shows a countermeasure against such attacks. The second contribution (Section 4) highlights side-channel attacks that have not been mentioned in previous work; these focus on the Goppa polynomial. As an aside, we also highlight in Appendix A noteworthy issues relating to the constant weight encoder; these are independent of our main contribution. We posit that in combination, these results are a step toward the security of both schemes on embedded platforms, and therefore toward the wider goal of usable post-quantum cryptography.

## 2 Background

Binary, irreducible CGCs represent the only code family for which the McEliece and Niederreiter Public-Key Cryptosystems (PKCs) have resisted cryptanalysis. Other subclasses of algebraic geometric codes have been considered, but failed to provide the necessary strength; see for example [17] or [21]. A binary, irreducible CGC with parameters $(n, k, d)$ is defined by an irreducible polynomial $G(X) \in \mathbb{F}_{2^m}[X]$ with $\deg(G(X)) = t = \lfloor \frac{d}{2} \rfloor$ (subsequently called the “Goppa polynomial”), and the code support $\mathbb{F}_{2^m} \supseteq \mathcal{L} = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\}$; such a parametrisation can correct at least $t$ bit errors. The defining equation is

$$\sum_{j=0}^{n-1} \frac{c_j}{X - \gamma_j} \equiv 0 \bmod G(X),$$

with $c_j$ denoting the j-th bit of the codeword $\mathbf{c} \in \mathbb{F}_2^n$, which allows derivation of the check matrix $\mathbf{H} \in \mathbb{F}_{2^m}^{t \times n}$. Based on this, the PKCs themselves can be defined as follows:

**Definition 1 (McEliece PKC).** *Let $\mathcal{C}(\mathcal{L}, G(X))$ be a binary, irreducible CGC. Furthermore, let*

$$\begin{array}{lll}
\mathbf{G} & \in \mathbb{F}_2^{k\times n} & \textit{be the generator matrix,} \\
\mathbf{S} & \in \mathbb{F}_2^{k\times k} & \textit{be a random, dense, non-singular scrambler matrix,} \\
\mathbf{Q} & \in \mathbb{F}_2^{n\times n} & \textit{be a random permutation matrix, and} \\
\mathbf{G}' & := \mathbf{SGQ} & \textit{be the hidden generator matrix.}
\end{array}$$

*The public and private keys are given by:*

$$\mathsf{PK}_{\mathsf{McE}} = \{n, k, t, \mathbf{G}'\} \qquad \textit{and} \qquad \mathsf{SK}_{\mathsf{McE}} = \{\mathcal{L}, G(X), \mathbf{H}, \mathbf{S}, \mathbf{Q}\}$$

*Let $\mathbf{e} \in \mathbb{F}_2^n$ be a random vector of weight $t$. Encryption of a data word $\mathbf{a} \in \mathbb{F}_2^k$ into a ciphertext $\mathbf{v}$ is given by*

$$\mathsf{enc}_{\mathsf{PK}_{\mathsf{McE}}}: \ \mathbf{v} \leftarrow \mathbf{aG}' + \mathbf{e},$$

*while decryption is given by*

$$\mathsf{dec}_{\mathsf{SK}_{\mathsf{McE}}}: \ \mathbf{a} \leftarrow \mathsf{decode}(\mathbf{vQ}^{-1})\mathbf{S}^{-1}.$$

**Definition 2 (Niederreiter PKC).** *Let $\mathcal{C}(\mathcal{L}, G(X))$ be a binary, irreducible CGC. Furthermore, let*

$$\begin{array}{lll}
\mathbf{H} & \in \mathbb{F}_{2^m}^{t\times n} & \textit{be the check matrix} \\
\mathbf{S} & \in \mathbb{F}_2^{mt\times mt} & \textit{be a random, dense non-singular scrambler matrix} \\
\mathbf{Q} & \in \mathbb{F}_2^{n\times n} & \textit{be a random permutation matrix, and} \\
\mathbf{H}' & := \mathbf{SHQ} & \textit{be the hidden check matrix denoted as } \mathbb{F}_2^{tm\times n} \textit{ matrix.}
\end{array}$$

*The public and private keys are given by:*

$$\mathsf{PK}_{\mathsf{Nie}} = \{n, k, t, \mathbf{H}'\} \qquad \textit{and} \qquad \mathsf{SK}_{\mathsf{Nie}} = \{\mathcal{L}, G(X), \mathbf{H}, \mathbf{S}, \mathbf{Q}\}$$

*Let the constant weight encoder* cw_encode *be a bijective, efficiently computable and efficiently invertible mapping*

$$\mathsf{cw\_encode}: \mathcal{A} \mapsto \{\mathbf{e} \in \mathbb{F}_2^n, \mathsf{weight}(\mathbf{e}) = t\}$$

*where $\mathcal{A}$ is the data space and* $\mathsf{weight}(\mathbf{e})$ *denotes the Hamming weight of* $\mathbf{e}$. *Encryption of such encoded data into a ciphertext $\mathbf{s}' \in \mathbb{F}_2^{mt}$ is given by*

$$\mathsf{enc}_{\mathsf{PK}_{\mathsf{Nie}}}: \ \mathbf{s}' \leftarrow \mathbf{H}'\mathbf{e}^\top.$$

*Decryption is given by*

$$\mathsf{dec}_{\mathsf{SK}_{\mathsf{Nie}}}: \ \mathbf{e} \leftarrow \mathbf{Q}^{-1}\mathsf{computeError}(\mathbf{S}^{-1}\mathbf{s}')$$

*whereafter $\mathbf{e}$ has to be decoded into plaintext using the inverse of the constant weight encoder,* $\mathsf{cw\_decode} := (\mathsf{cw\_encode})^{-1}$.

**Algorithm 1.** McEliece decryption with Patterson's algorithm.

```
INPUT: The sense word $\mathbf{v} \in \mathbb{F}_2^n$, matrices $(\mathbf{Q}\mathcal{L})$, $(\mathbf{Q}^{-1}\mathbf{H})$, $\mathbf{G}'^{-1}$ and the Goppa polynomial $G(X)$.
OUTPUT: The clear text $\mathbf{a}$

1.  $S(X) \leftarrow \mathbf{v}(\mathbf{Q}^{-1}\mathbf{H})^\top$
2.  $S'(X) \leftarrow S^{-1}(X) \bmod G(X)$                      [For example with the EEA.]
3.  if $S'(X) = X$ then $u(X) \leftarrow S'(X)$
4.  else
5.      $S'(X) \leftarrow \sqrt{S'(X) + X}^{\bmod G(X)}$          [Polynomial square root $\bmod G(X)$]
6.      $[\mathrm{x}(X), \mathrm{y}(X)] \leftarrow$ EEA_Decode $(G(X), S'(X), \lceil t/2 \rceil)$    [EEA_Decode is given in Algorithm 5]
7.      $u(X) \leftarrow \mathrm{x}^2(X) + X\mathrm{y}^2(X)$
8.  for $\gamma_i \in (\mathbf{Q}\mathcal{L})$                          [$i = 0, \ldots, n-1$]
9.      if $u(\gamma_i) = 0$ then $\mathbf{v} \leftarrow$ toggleBit $(\mathbf{v}, i)$
10. return $\mathbf{a}' \leftarrow \mathbf{v}\mathbf{G}'^{-1}$
```

In the following sections we focus on McEliece decryption, making use of the well known Patterson Algorithm [12] to decode the CGC as described in Algorithm 1. For the Niederreiter PKC, the same algorithm can be trivially adapted to suit. The initial permutation is avoided by using a prepermuted check matrix and a prepermuted code support. This results in a more (time- and memory-) efficient implementation and reduces the risk of side-channel attack by eliminating the associated leakage.

We implemented this algorithm on Linux AMD64 and Mac PowerPC architectures using a specialised library for arithmetic in small binary fields: it implements multiplication and exponentiation of field elements using logarithmic and anti-logarithmic Look-Up Tables (LUTs) and supports polynomial computations of these fields as well as vectors and matrices over these fields and the base field. NTL [11] was used to support number theoretic operations during the key generation and for testing purposes.

We have focused on the following parametrisations with security estimates based on [2]:

- $m = 11$ and $(n, k, d) = (2048, 1751, 55)$ provides 80-bit security,
- $m = 12$ and $(n, k, d) = (2960, 2288, 113)$ provides 128-bit security, and
- $m = 13$ and $(n, k, d) = (6624, 5129, 231)$ provides 256-bit security.

To allow easier comparison with previous work, we also consider the "classic" case $m = 10$ and $(n, k, d) = (1024, 524, 101)$ which should be considered insecure.

## 3 Timing Related Side-Channels

Within Algorithm 1 there are basically four exploitable side-channels relating to variation in execution time:

- The first side-channel was published in [19] and concerns Line 9. The side-channel itself will be discussed in Section 3.1 where we introduce the first effective countermeasure, the *non-support*.

- The number of rounds performed by the **EEA_Decode** Algorithm, as used in Line 6, depends on the error weight; attacks based on this fact were presented in both [16] and [20]. In Section 3.2 we discuss why the *non-support* countermeasure fails against such attacks, and why Line 2 does leak information through a similar side-channel.
- Section 3.3 discusses the third and fourth side-channels which relate to the matrix operation in Line 10, and the impact of cache behaviour on any LUT-based realisation of finite field arithmetic.

## 3.1 Polynomial Evaluation

A simple, yet very successful timing attack on McEliece encryption is given by Strenzke et al. [19]. It is based on the observation that the time needed for the error computation step is directly related to the error weight, and this dominates the total time required for decryption. The attack is basically a reaction attack that uses the timing difference to obtain the error positions in a ciphertext. When these positions are known to an attacker, correcting the error and recovering the message becomes easy. The secret key is not compromised.

In Algorithm 1, the error computation is equal to evaluating the error locator polynomial $u(X)$ on all points of the code support which, by definition of $u(X)$, is equal to factoring $u(X)$. This is usually done using a Horner scheme, resulting in a time estimation of

$$t_{\mathsf{EVAL}} = |\mathcal{L}| \deg(u(X))(t_{\mathsf{MUL}} + t_{\mathsf{ADD}}).$$

The correlation between error weight and decryption time using a straight forward implementation of Patterson's algorithm is shown by the plots *not* marked "hardened" in Figure 1.

Based on the suggestions of [19] we have been able to improve the timing attack with a setup stage (Algorithm 2) that profiles the algorithm for all correctable error weights, and an iterative process (Algorithm 3) that approximates the random error vector by using the previously collected timing profiles to measure the success of each iteration. Thus we improve the already reasonable success probability of 50% of the original attack to obtain a success probability greater than 99%: in fact we recovered the correct error vectors for 300 out of 300 randomly chosen ciphertexts.

However, we propose the "non-support" as a simple, efficient and effective countermeasure against this attack. It is defined as follows:

**Definition 3.** *Let $\mathcal{L} \subseteq \mathbb{F}_{2^m}$ be the code support. The non-support is defined as*

$$\overline{\mathcal{L}} = \mathbb{F}_{2^m} \setminus \mathcal{L}$$

*if and only if $\mathcal{L} \subsetneq \mathbb{F}_{2^m}$, as*

$$\overline{\mathcal{L}} = \mathbb{F}_{2^{m'}} \setminus \mathcal{L}$$

*with $m' \geq m+1$ if and only if $|\mathcal{L}| = 2^m$ and a free choice of $m'$ is possible or as*

$$\overline{\mathcal{L}} = \mathbb{F}_{2^{xm}} \setminus \mathbb{F}_{2^m}$$

*with $x \geq 2$ if and only if $|\mathcal{L}| = 2^m$ and $m$ is fixed. In this last case, the error computation has to be done in the extension field $\mathbb{F}_{(2^m)^x}$.*

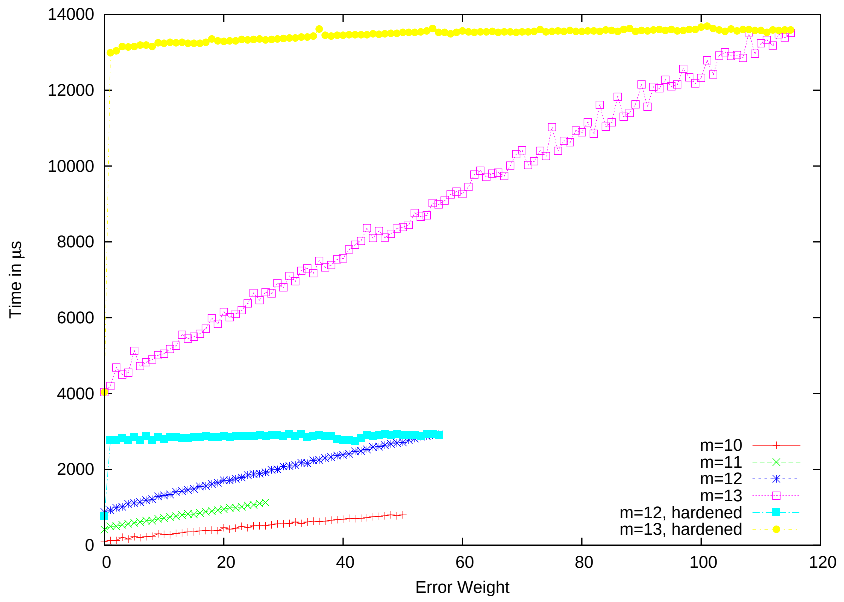

**Fig. 1.** Average decryption time for given (correctable) error weights on a Linux AMD64 machine. The code parameters are $m = 10$ and $(1024, 524, 101)$, $m = 11$ and $(2048, 1751, 55)$, $m = 12$ and $(2960, 2288, 113)$, $m = 13$ and $(6624, 5129, 231)$. Measurements for a hardened decryption (using Algorithm 4) have been added for the cases $m = 12$ and $m = 13$.

Clearly the last case will incur considerable overhead, but from a theoretic point of view it can be avoided by choosing $m' \geq m + 1$ and from a practical point of view one has to wonder whether the decision to enforce $m' = m$ is justified compared to the overhead of doing computations in $\mathbb{F}_{(2^m)^x}$.

The non-support has been defined such that any root added to the error locator polynomial $u(X)$ from the non-support will not affect the error computation. Using that, we are able to compute a fake error locator $\overline{u(X)}$ and a hardened error locator $u'(X)$ which has $deg(u'(X)) = t$ no matter what the input is; this is illustrated by Algorithm 4. The success of the proposed hardening is visible in the plots marked "hardened" in Figure 1. To mount our improved timing attack successfully, we have to correct $\lceil \frac{t}{2} \rceil$ error bits in the first iteration. Considering that there are no reliable timing differences for the error weights $t - \epsilon \ldots t$ in the hardened cases, using the attack on a hardened decryption makes no sense. The sharp decline of the hardened cases for $\mathsf{weight}(\mathbf{e}) = 0$ is easily explained by the fact that in this case the decryption algorithm breaks off before the locator computation step; an attacker who is able to remove all errors on the first run has broken the PKC anyway. Furthermore, the plots show that the hardening is inexpensive compared to the entire decryption which, in untampered scenarios, always has $deg(u(X)) = t$.

The small spikes and the slight decline for

$$0 < \mathsf{weight}(\mathbf{e}) < \frac{t}{2}$$

are explained in the Sections 3.2 and 3.3. However, some system dependent noise could not be avoided during the measurements and adds to the spikes as well.

**Algorithm 2.** Setup stage for the timing attack against the McEliece decryption.

INPUT: The McEliece decryption module under attack with public key $\{n, k, t, \mathbf{G}'\}$, number of samples taken per error weight ($numOfSamples$)
OUTPUT: An array $\mathbf{E}$ containing the average decryption time per error weight.

```
1.  for $i = 0$ to $t$
2.      $\mathbf{E}[i] \leftarrow 0$
3.      for $j = 0$ to $numOfSamples$
4.          $\mathbf{E}[i] \leftarrow \mathbf{E}[i] +$ time ( decrypt ( randCodeWord $(\mathbf{G}')\oplus$ randErrorOfWeight $(i)$))
5.  return $(\mathbf{E}/numOfSamples)$
```

**Algorithm 3.** Improved timing attack against the McEliece decryption.

INPUT: A senseword $\mathbf{v}$ of length $n$ containing $t$ errors and an array $\mathbf{E}$ computed by Algorithm 2
OUTPUT: The most likely data word $\tilde{\mathbf{a}}$

```
1.  $\tilde{\mathbf{v}} \leftarrow \mathbf{v}$; $\tilde{t} \leftarrow t$
2.  while $0 \neq \tilde{t}$                                [Repeat until current error weight $= 0$.]
3.      for $i = 0$ to $n-1$                                [Timing Attack.]
4.          $\hat{\mathbf{v}} \leftarrow$ toggleBit $(\tilde{\mathbf{v}}, i)$
5.          $\mathbf{time}[i] \leftarrow$ time ( decrypt $(\hat{\mathbf{v}})$)
6.      $\mathbf{places} \leftarrow$ selectIndicesOfSmallestValues $(\mathbf{time}, \tilde{t})$
7.      $\tilde{\mathbf{v}} \leftarrow$ toggleBits $(\tilde{\mathbf{v}}, \mathbf{places})$
8.      $time \leftarrow$ time ( decrypt $(\tilde{\mathbf{v}})$)             [Estimate current error weight.]
9.      for $i = 1$ to $t$ step $2$                         [Odd error weights can be ignored at this point.]
10.         if $time \leq \mathbf{E}[i]$
11.             $\tilde{t} \leftarrow i - 1$; break
12. return $\tilde{\mathbf{a}} \leftarrow \tilde{\mathbf{v}}(\mathbf{QGS})^{-1}$
```

## 3.2 The Extended Euclidean Algorithm (EEA)

In [16] and [20] the authors observe and explain that the number of rounds spent in the**EEA_Decode** algorithm depends on the error weight and use this difference in the timing to mount a reaction attack similar to [19]. As countermeasure they propose to raise the degree of the polynomial within the**EEA_Decode** artificially, but they do not detail their proposed countermeasure exactly; in particular they do not describe how this approach would avoid effecting the evaluation of the error locator polynomial.

Again, the *non-support* would be beneficial in achieving an execution time independent from the error weight. A *non check matrix* $\overline{\mathbf{H}}$ can be computed in advance, similar to the check matrix, using the elements of the non-support instead of the elements of the code support. If now a trivially computable function $f$ with

$$f: \ \mathsf{weight}(\mathbf{e}) = f(S(X))$$

**Algorithm 4.** Hardened McEliece decryption with Patterson's algorithm.

```
INPUT: Senseword $\mathbf{v}$, secret key $\mathsf{SK}_{\mathsf{McE}}$, non-support $\overline{\mathcal{L}}$.
OUTPUT: Plaintext $\mathbf{a}$.

1.  ...                                                  [Syndrome & locator computation, see Alg. 1.]
2.  $\mathbf{u}[\mathtt{TRUE}] \leftarrow u(X), \mathbf{u}[\mathtt{FALSE}] \leftarrow 1$        $\left[\mathbf{u}[\mathtt{TRUE}] := u'(X); \mathbf{u}[\mathtt{FALSE}] := \overline{u(X)}\right]$
3.  for $i = 0$ to $t-1$
4.      $\overline{\gamma_i} \leftarrow$ chooseElement $(\overline{\mathcal{L}})$
5.      boolean $b \leftarrow \deg(u'(X)) \leq i$            [e.g. boolean := $\{\mathtt{TRUE}, \mathtt{FALSE}\} \mapsto \{1, 0\}$]
6.      $\mathbf{u}[b] \leftarrow \mathbf{u}[b](X + \overline{\gamma_i})$
7.  $u'(X) \leftarrow \mathbf{u}[\mathtt{TRUE}]$
8.  for $i = 0$ to $n-1$                                 [Error computation.]
9.      if $u'(\gamma_i) = 0$ then $\mathbf{v} \leftarrow$ toggleBit $(\mathbf{v}, i)$       [Horner scheme.]
10. return $\mathbf{a}' \leftarrow \mathbf{v}(\mathbf{SGQ})^{-1}$
```

exists, we would be able to add $t - f(S(X))$ rows of the *non check matrix* to the syndrome (in a way similar to Algorithm 4) before any other computations are performed on the syndrome. This would counter all timing side-channels originating from the error computation without incurring any other costs. However, we have not been able to find such a function $f$ and thus lack a criteria to decide how many rows need to be added. We suggest this is an open problem that should be addressed in further research.

But even before **EEA_Decode** is invoked in Line 6 of Algorithm 1, another invocation of the EEA is performed for the polynomial inversion in Line 2; previous work has not investigated this invocation at all, but clearly one has to consider it as a potential timing side-channels as well. For

$$\mathsf{weight}(\mathbf{e}) < \frac{t}{2}$$

an interesting situation arises: the error could be corrected by a more generic algorithm, e.g., by the algorithm presented in [18] which would replace the inversion with an invocation of

$$\textbf{EEA\_Decode}(G(X), S(X), \lceil t/4 \rceil).$$

It is obvious that for this case we have to cope with the resulting timing side-channel as well. For

$$\mathsf{weight}(\mathbf{e}) > \frac{t}{2}$$

we have no theoretic explanation so far but experimental results show the number of rounds to be constant with a very small number of exceptions. This means, an attacker would have to correct $t/2$ or more errors before being able to use this sidechannel which is infeasable. Further, the exceptions observed make an iterative approximation as described in the Section 3.1 less reliable. Our experimental results are shown in Figure 2.

## 3.3 Other Timing Related Side-Channels

Finally, we remark briefly on two further source of timing variation and hence potentially exploitable side-channels. We stress that these have not been exploited via real attacks

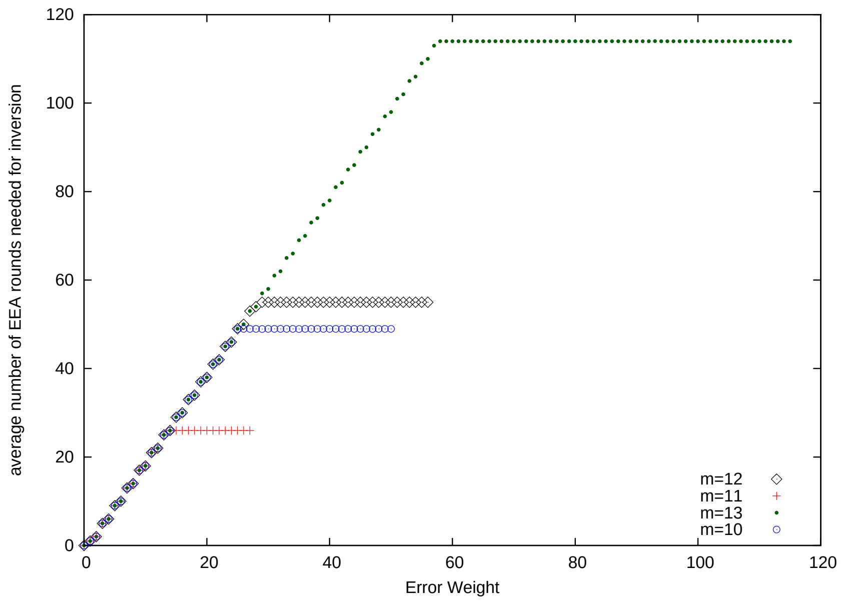

**Fig. 2.** Average number of rounds needed for the extended euclidean algorithm in the inversion step of Algorithm 1. Averaged on 2000 random samples.

on our experimental platforms, but rather we suggest that doing so represents an open problem that should be addressed in further research.

## The Matrix Operation

The matrix operation in Line 10 of Algorithm 1 may influence execution time and thereby represent a timing side-channel. Assume that all other timing side-channels relating to error correction have been avoided; imagine the attacker modifies one bit of the ciphertext $\mathbf{v}$, and asks the device under attack to decrypt the result $\mathbf{v}'$. If the modified bit is an error bit, the error correction computes the correct code word and the operand of the final matrix multiplication is the same; if it was not an error bit, the error correction computes a wrong code word with potentially different Hamming weight. This will change the time required for the matrix multiplication and allow the attacker to discover the error positions (and a few false positives) unless proper measures are taken to ensure constant time execution. Two possible solutions are:

1. Use of a window-algorithm for matrix multiplication. This increases the amount of memory required by a factor $2^w$, but is faster by a factor $w$ if $w$ denotes the window size. The possibility of a "zero window" is $1/(2^w)$ and the numbers of zero windows would have to differ between the correct and a wrong code word so that the attacker sees the difference between both.
2. Alternatively, all rows of the retrieval matrix are added but the unneeded rows are added to a fake plaintext instead. This means that always the worst case time is needed but the additional memory is limited to $k$ bits.

**Cache Behaviour**

In the case of the McEliece and Niederreiter PKCs, all performance-oriented implementations use logarithmic and anti-logarithmic LUTs to realise finite field arithmetic; for the parameters used, various related quantities are given in Table 1. The chance of retaining the entire working set in (a reasonably sized) cache memory is low: this implies that cache interference based on access to the LUTs is inevitable, and that the variation in execution due to the resulting cache-misses leaks information. We suggest that this affords an attack the possibility of recovering information about operands used in finite field operations by virtue of their use as addresses in LUT access.

| | $m = 10$ | | $m = 11$ | | $m = 12$ | | $m = 13$ | |
|---|---|---|---|---|---|---|---|---|
| | 10 bpE | 16 bpE | 11 bpE | 16 bp | 12 bpE | 16 bpE | 13 bpE | 16 bpE |
| log table | 1280B | 2048B | 2816B | 4096B | 6144B | 8192B | 13312B | 16384B |
| log & anti-log tables | 2560B | 4096B | 5632B | 8192B | 12288B | 16384B | 26624B | 32768B |
| **H** | 64000B | 102400B | 76032B | 110592B | 248640B | 331520B | ≈ 1.18MB | ≈ 1.45MB |
| $G(X)$ | 63.75B | 102B | 38.5B | 56B | 85.5B | 114B | 188.5B | 232B |
| $\sqrt{X}^{\bmod G(X)}$ | 62.5B | 100B | 37.125B | 54B | 84B | 112B | 186.875B | 230B |

**Table 1.** Sizes for log & anti-log LUTs, and the two required polynomials, measured in bytes ("bpE" stands for "bits per (field) element", 16 is the size of an `unsigned short` type on our experimental platforms).

## 4 Side-Channel Attacks on the Goppa Polynomial

To consider direct side-channel leaks from Patterson's algorithm on the Goppa polynomial, one has to look at the locator computation as shown in Algorithm 1. The Goppa polynomial appears implicitly in Line 5 and explicitly in Lines 2 and 6 (both of which represent leakages from the EEA).

Note that these leakages impact on both PKCs; only minor differences in the setup of the attack need to be considered for the attack to accommodate both.

**Implicit Leakage from the Square Root Computation**

The preferred method to compute $\sqrt{S'(X) + X}^{\bmod G(X)}$ has been given by Huber in [6]. (For a short description see Appendix B.) Here the Goppa polynomial is not involved directly, but rather during the square root computation a multiplication with the polynomial

$$\sqrt{X}^{\bmod G(X)},$$

which is constant and unique for given $G(X)$, has to be performed. If this constant leaks from the polynomial multiplication, the attacker can compute

$$\mathtt{y}(X) = (\sqrt{X}^{\bmod G(X)})^2 + X = \mathtt{z}(X)G(X)$$

and obtains $G(X)$ by factoring $\mathtt{y}(X)$. Having $t \leq \deg(\mathtt{y}(X)) < 2t$ and $G(X)$ irreducible with $\deg(G(X)) = t$ this is no problem. The attack on Huber's method will be simplified if the attacker can choose a syndrome such that

$$S'(X) \leftarrow S^{-1}(X) \bmod G(X)$$

has all coefficients $S'_{2i+1}$ set to zero.

**Algorithm 5. EEA_Decode**

INPUT: $\mathtt{y}(X), \mathtt{z}(X)$ having $\deg(\mathtt{y}(X)) \geq \deg(\mathtt{z}(X))$, bound $b \in \mathbb{N}$; use $b = 1$ for **EEA**
OUTPUT: $\mathtt{x}_E(X)$, $\mathtt{z}_E(X)$ such that $\mathtt{x}_E(X) \equiv \mathtt{z}_E(X)\mathtt{z}(X) \bmod \mathtt{y}(X)$ and $\mathtt{x}_E(X)$ having maximal degree with $deg(\mathtt{x}_E(X)) < b$

```
1.  $\mathtt{x}_0(X) \leftarrow \mathtt{y}(X);\ \mathtt{x}_1(X) \leftarrow \mathtt{z}(X)$
    $\mathtt{z}_0(X) \leftarrow 0;\quad \mathtt{z}_1(X) \leftarrow 1$
2.  while $\deg(\mathtt{x}_{i+1}(X)) \geq b$ do                 [loop counter: $i = 1, \ldots$]
3.      $q(x) \leftarrow 0;\ \mathtt{x}_{i+1}(X) \leftarrow \mathtt{x}_{i-1}(X)$
        $l \leftarrow \deg(\mathtt{x}_i(X))$                     [quoRem: quotient $q(X)$,]
4.      for $j = \deg(\mathtt{x}_{i-1}(X))$ downto $l$         [$\mathtt{x}_{i+1}(X) \leftarrow \mathtt{x}_{i-1}(X) \bmod \mathtt{x}_i(X)$]
5.          $q_{j-l} \leftarrow$ coeff $(\mathtt{x}_{i+1}(X),\ j)$ / coeff $(\mathtt{x}_i(X),\ l)$
6.          $\mathtt{x}_{i+1}(X) \leftarrow \mathtt{x}_{i+1}(X) + q_{j-l}X^{j-l}\mathtt{x}_i(X)$
7.      $\mathtt{z}_{i+1}(X) \leftarrow \mathtt{z}_{i-1}(X) + q(X)\mathtt{z}_i(X)$
8.  return $(\mathtt{x}_E(X) \leftarrow \mathtt{x}_{i+1}(X), \mathtt{z}_E(X) \leftarrow \mathtt{z}_{i+1}(X))$
```

## EEA Leakage

A typical implementation of **EEA_Decode**, which can be seen as a generalisation of the EEA, is shown in Algorithm 5. In both cases of Patterson's algorithm, $\mathtt{x}_0(X)$ will be initialised with $G(X)$, and the Lines 5 and 6 during the first iteration of the while-loop will yield the most for the attacker. $\mathtt{x}_1(X)$ will be initialised either with $S(X)$ or $S'(X)$. Since the algorithmic properties are the same in both cases, we continue using $S(X)$ without loss of generality; the cases differ only in the effort necessary to choose an appropriate input polynomial. Ideally, the attacker would want to have

$$\mathtt{x}_1(X) = S(X) = \alpha \in \mathbb{F}_{2^m}^*$$

and circumvent (using a fault attack) the condition in Line 2 so that the while-loop can be observed. In that case, the for-loop would process all coefficients of the Goppa polynomial and Line 5 would be

$$q_{j-l} \leftarrow G_j\alpha^{-1}.$$

Furthermore, the attacker may assume $G_t = 1$ and will thus learn the value of $\alpha^{-1}$. However, the attacker might not be able to circumvent the condition of Line 2. In that case, the attacker would want to have:

$$\mathtt{x}_1(X) = S(X) = \alpha X + \beta, \qquad \alpha \in \mathbb{F}_{2^m}^*, \beta \in \mathbb{F}_{2^m}$$

Then the for-loop will only process the coefficients $G_t \ldots G_1$ in Line 5 and the $G_0$ coefficient has to be recovered either from observations on Line 6 or through a brute force search which is feasible since only $2^m - 1$ elements have to be searched. However, for syndrome polynomials of degree 1 Line 5 will not be as friendly as for the degree 0 case

$$q_{t-1} \leftarrow G_t \alpha^{-1} \ , \quad q_{t-2} \leftarrow (G_{t-1} + q_{t-1}\beta)\alpha^{-1} \ , \quad q_{t-3} \leftarrow (G_{t-2} + q_{t-2}\beta)\alpha^{-1} \ , \quad \ldots$$

With the same assumption that $G_t = 1$ is used, $\alpha^{-1}$ can be observed. However, $\beta$ has to be recovered either in Line 6 or a syndrome polynomial with $\beta = 0$ has to be chosen by the attacker leading to a significantly reduced number of possible syndrome polynomials. On the other hand, $\beta \neq 0$ leads to error propagation into all $q_{j<i}$ if a wrong $q_i$ has been assumed. This approach can be generalised to accommodate syndromes of degrees larger than 1 but incurs an increase in complexity and vulnerabilities to error propagation.

**Abilities of the Attacker**

To evaluate the difficulty of these attacks, one has to assess the abilities an attacker is required to have. To make this assessment, two reasonable assumptions on the implementation of field operations are given:

- Additions are a simple XOR of two elements and leak almost no data compared to multiplication.
- For multiplication, the log/anti-log LUTs are used which allows the operation to be implemented without branches. The use of LUTs implies that the attacker will try to identify which element of a LUT is accessed at a given moment.

Based on these assumptions, the attacker needs to have the following abilities:

- Learn the position of multiplication coefficients in the log/anti-log LUTs.
- To exploit the EEA leakage the attacker has to be able to learn the degree of $S(X)$ or $S'(X)$. Note that this might also be used to acquire information on the Niederreiter scrambler matrix. For the Niederreiter PKC we expect that a successful attacker tries to use the knowledge he acquires on the scrambler matrix to reduce the time needed to find low degree polynomials.

Furthermore, any fault induction abilities, either on the condition of the while-loop or on the coefficients of the polynomials $S(X)$ and $S'(X)$ reduce the attack complexity.

## 5 Conclusions

Development and security analysis of post-quantum cryptographic schemes is an active research area; the study of efficiency and physical security within the same context is vital if said schemes are to be deployed and used.

Modified decryption paths [7, 5] for the McEliece and Niederreiter PKCs optimise computational effort and already increase the complexity of side-channel attack. However, by introducing the concept of non-support we have devised an efficient countermeasure against specific timing attacks and hence closed the gap left by [19] and [16]. Furthermore, our analysis of vulnerabilities in the error locator computation outlines the research questions that have to be answered for secure, embedded implementations.

Finally, our results in appendix A demonstrate that it is not sufficient to secure the encryption in its purest form when substantial data may be leaked from necessary, but cryptographically irrelevant, preprocessing steps such as constant weight encoders.

## References

1. D.J. Bernstein, T. Lange and C. Peters. *Attacking and defending the McEliece cryptosystem.* In: Proceedings of PQCrypto 2008, LNCS 5299, pages 31–46, 2008. See also: Cryptology ePrint Archive, Report 2008/318, 2008. URL: `http://eprint.iacr.org/2008/318.pdf`
2. B. Biswas and N. Sendrier. *McEliece Cryptosystem Implementation: Theory and Practice.* In: Proceedings of PQCrypto 2008, LNCS 5299, pages 47–62, Springer-Verlag Berlin Heidelberg, October 2008.
3. N. Courtois, M. Finiasz and N. Sendrier. *How to achieve a McEliece-based Digital Signature Scheme.* Cryptology ePrint Archive, Report 2001/010, 2001.
URL: `http://eprint.iacr.org/2001/010.pdf`
4. T. Eisenbarth, T. Güneysu, S. Heyse and C. Paar. *MicroEliece: McEliece for Embedded Devices.* In *Cryptographic Hardware and Embedded Systems (CHES)*, Springer-Verlag LNCS 5747, pages 49–64, 2009.
5. S. Heyse, A. Moradi, C. Paar. *Practical Power Analysis Attacks on Software Implementations of McEliece.* To appear in *Post-Quantum Cryptography 2010 (PQCrypto 2010)*, Springer-Verlag LNCS 6061.
6. K. Huber. *Note on decoding binary Goppa codes* In *Electronics Letters*, vol. 32, no. 2, pages 102–103, 18 Jan 1996
URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=490862&isnumber=10460`
7. S. Hoerder. *Explicit Computational Aspects of McEliece Encryption Schemes.* Diploma Thesis. Ruhr-Universität Bochum, August 2009. Made available to the authors in private communication.
8. R.J. McEliece. *A public-key cryptosystem based on algebraic coding theory.* Jet Propulsion Laboratory DSN Progress Report 42–44, January and February 1978, pages 114-116.
URL: `http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF`.
9. R. Merkle. *A certified digital signature.* In Advances in Cryptology – CRYPTO '89, LNCS 1462, pages 218–238. Springer, 1989.
10. H. Niederreiter. *Knapsack-Type Cryptosystems and Algebraic Coding Theory. Problems of Control and Information Theory* (Problemy Upravlenija i Teorii Informacii) **15**, 1986, pages 159–166.
11. V. Shoup. *NTL - a library for doing numbery theory, v. 5.4.1.* May 2007, http://www.shoup.net/ntl/
12. N. Patterson. *The algebraic decoding of Goppa codes. IEEE Transactions on Information Theory* **21/2**, 1975, pages 203–207.
13. J. Proos and C. Zalka. *Shor's discrete logarithm quantum algorithm for elliptic curves.* Quantum Information and Computation, Vol. 3, pages 317–344, 2003.
14. N. Sendrier. *Encoding Information into Constant Weight Words.* In Proceedings of the 2005 IEEE International Symposium on Information Theory, Adelaide, pages 435-438, Springer-Verlag Berlin Heidelberg, September 2005
15. P.W. Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer.* In *Foundations of Computer Science*, IEEE Computer Society Press, pages 124–134, 1994. Extended version: SIAM Journal on Computing, Vol. 26, pages 1484–1509, 1997.
16. A. Shoufan, F. Strenzke, H.G. Molter and M. Stöttinger. *A Timing Attack Against Patterson Algorithm in the McEliece PKC.* In ICISC 2009
17. V.M. Sidel'nikov and S.O. Shestakov. *On Insecurity of Cryptosystems Based on generalized Reed-Solomon Codes.* Discrete Mathematics and Applications 2/4 1992, pages 439–444.
18. Y. Sugiyama, M. Kasahara, S. Hirasawa, T. Namekawa. *A Method for Solving Key Equation for Decoding Goppa Codes.* In *Information and Control* Vol. 27, 1975, pages 87–99
19. F. Strenzke, E. Tews, H.G. Molter, R. Overbeck and A. Shoufan. *Side Channels in the McEliece PKC.* In Proceedings of PQCrypto 2008, LNCS 5299, pages 216–229, Springer-Verlag Berlin Heidelberg, October 2008.

20. F. Strenzke. *A Timing Attack against the secret Permutation in the McEliece PKC.* To appear in the Proceedings of PQCrypto 2010.
21. V. Gauthier Umaña and G. Leander. *Practical Key Recovery Attacks On Two McEliece Variants.* URL: `http://eprint.iacr.org/2009/509.pdf`

## A Implementation Issues: Constant Weight Encoding

When we implemented the Niederreiter encryption (see Algorithm 6) on an ARM7TDMI-based LPC2124 microprocessor we discovered two issues related to the constant weight encoder which are noteworthy but not part of our main contribution. The constrained nature of this platform meant that we focused on the first of the previous parametrisations (i.e., 80-bit) only; given the 16 kB RAM and 256 kB flash memory available, the 74 kB public-key was stored in flash memory.

The instantiation of cw_encode, the constant weight encoder, is important in both PKCs: it is inherent within the Niederreiter PKC, and can optionally be used to transform the output of a random number generator into random error words (of weight $t$) within the McEliece PKC. The cryptanalytic security of neither PKC depends on the constant weight encoder, but the construct *should* be considered within the context of side-channel attack. As such, the design of a constant weight encoder should ideally fulfil (at least) the following requirements:

1. efficiency of computation,
2. efficiency of encoding, and
3. minimisation of side-channel leakage.

In relation to the second requirement, a constant weight encoder can encode at most $\log_2 \binom{n}{t}$ input bits into one word. In the McEliece PKC the encoding efficiency has to be large enough to prevent a complete enumeration of error words, and in the Niederreiter PKC the encoding efficiency affects the effective bandwidth.

To our knowledge, there currently seems to be no constant weight encoder that satisfies all requirements. We selected the Sendrier constant weight encoder (see [14] and Algorithm 7) which performs well in both efficiency categories, but uses variable length inputs and (for the 80-bit parametrisation) leaks some bits. Using a template attack against our microprocessor implementation, we were able to obtain (albeit with approximate placement) the values of the bits read in Line 7 of Algorithm 7. Using a simulation with $10^7$ random messages, we established that this instruction reads on average $\approx 26.6\%$ of the message with the minimum being $\approx 13.9\%$ and the maximum being $\approx 53.0\%$. In the absolute worst case (the “all one” message which was not part of the random sample of messages), 100% of the message are revealed by the timing attack.

Sendrier’s constant weight encoder is highly recursive in the original version. Due to the memory constraints on our platform, stack size in particular, we reformulated the algorithm using the recursion-free approach presented in Algorithm 7.

## B Huber’s Polynomial Modular Square Root Computation Method

For those not familiar with Huber’s method (see [6]), we give a short description of it: It requires that we precompute $\sqrt{X}^{\bmod G(X)}$ for an irreducible Goppa polynomial $G(X)$. To

**Algorithm 6.** niederreiterEnc, Niederreiter encryption.

INPUT: $\mathrm{PK}_{\mathrm{Nie}} = \{n, k, t, \mathbf{H}'\}$, a binary data stream, precomputed $best$ (see Algorithm 7)
OUTPUT: A ciphertext $\mathrm{s}'$

```
1.  $\Delta \leftarrow$ cw_encode$(n, t, 0, best)$          [$\Delta[i]$ = number of zeros in e before next '1'.]
2.  $\mathrm{s}' \leftarrow$ column( $\mathbf{H}', \Delta[0]$ )
3.  for $i$ from $1$ to $t-1$
4.      $\Delta[i] \leftarrow \Delta[i-1] + \Delta[i] + 1$
5.      $\mathrm{s}' \leftarrow \mathrm{s}' \oplus$ column( $\mathbf{H}', \Delta[i]$ )
6.  return $\mathrm{s}'$
```

**Algorithm 7.** sendrierCWE, recursion-free Sendrier constant weight encoding.

INPUT: Bits in vector $\mathtt{x} = n$, weight $\mathtt{y} = t$, current distance $\delta = 0$, precomputed
$best = \left\{1, 1-\frac{1}{2}, 1-\frac{1}{\sqrt{2}}, \ldots, 1-\frac{1}{\sqrt[t]{2}}\right\}$, an input bit stream
OUTPUT: Vector $\Delta$ of y distances

```
1.  while $\mathtt{y} > 0$
2.      if $\mathtt{x} \leq \mathtt{y}$
3.          $\Delta[t-\mathtt{y}] \leftarrow \delta$
4.          $\mathtt{x} \leftarrow \mathtt{x}-1$; $\mathtt{y} \leftarrow \mathtt{y}-1$; $\delta \leftarrow 0$; continue     [i.e. start next loop iteration immediately.]
5.      $\mathtt{z} \leftarrow$ round$\Big( \big(\mathtt{x} - \frac{\mathtt{y}-1}{2}\big) * best[\mathtt{y}] \Big)$
6.      if $\mathtt{z} = 0$ then $\mathtt{z} \leftarrow 1$          [Avoid infinite loops.]
7.      if ( readBits$(1) = 1$ )          [Reads one bit; template based SPA reveals the value.]
8.          $\mathtt{x} \leftarrow \mathtt{x} - \mathtt{z}$; $\delta \leftarrow \delta + \mathtt{z}$; continue
9.      $\mathtt{l} \leftarrow \lceil \log_2(\mathtt{z}) \rceil$          [Lines 9 to 11 = subfunction decodefd from [14].]
10.     $\delta_{tmp} \leftarrow$ readBits$(\mathtt{l}-1)$          [$\delta_{tmp} \leftarrow 0$ if and only if $\mathtt{l} = 1$]
11.     if $\delta_{tmp} \geq (2^{\mathtt{l}} - \mathtt{z})$ then $\delta_{tmp} \leftarrow 2\delta_{tmp} +$ readBits$(1) - (2^{\mathtt{l}} - \mathtt{z})$
12.     $\Delta[t-\mathtt{y}] \leftarrow \delta + \delta_{tmp}$
13.     $\mathtt{x} \leftarrow \mathtt{x} - \delta_{tmp} - 1$; $\mathtt{y} \leftarrow \mathtt{y} - 1$; $\delta \leftarrow 0$
14. return $\Delta$
```

do so we split $G(X)$ into polynomials $g_1(X)$ and $g_2(X)$

$$G(X) = \underbrace{\left(g_0 + g_2X^2 + g_4X^4 + g_6X^6 + \ldots\right)}_{g_1(X)} + X\underbrace{\left(g_1 + g_3X^2 + g_5X^4 + \ldots\right)}_{g_2(X)}$$

such that the computation of $\sqrt{g_1(X)}$ and $\sqrt{g_2(X)}$ is reduced to computing the square roots of the coefficients in $\mathbb{F}_{2^m}$and obtain

$$\begin{aligned} g_1(X) &\equiv Xg_2(X) \bmod G(X) \\ \sqrt{g_1(X)} &\equiv \sqrt{X}^{\bmod G(X)}\sqrt{g_2(X)} \bmod G(X) \\ \sqrt{X}^{\bmod G(X)} &\equiv \sqrt{g_1(X)}\left(\sqrt{g_2(X)}\right)^{-1} \bmod G(X) \end{aligned}$$

using $\gcd(G(X), \sqrt{g_2(X)}) = 1$. (Here we profit from $G(X)$ being irreducible.)

To compute $\sqrt{S'(X) + X}^{\bmod G(X)}$ we split $\tilde{S}(X) = S'(X) + X$ into polynomials $\tilde{s}_1(X)$ and $\tilde{s}_2(X)$ (just as we split the Goppa polynomial) such that we can easily compute their square roots. The final result follows:

$$\sqrt{\tilde{S}(X)}^{\bmod G(X)} \equiv \left(\sqrt{\tilde{s}_1(X)} + \sqrt{X}^{\bmod G(X)}\sqrt{\tilde{s}_2(X)}\right) \bmod G(X)$$

Thus we get the square root $\sqrt{S'(X) + X}^{\bmod G(X)}$ at the cost of performing $t$ square roots in $\mathbb{F}_{2^m}$ which can be done (due to the small $m$) very efficiently with a LUT similar to the logarithmic LUTs we already mentioned and a polynomial multiplication modulo $G(X)$.