Leakage Measurement Tool of McEliece PKC Calculator

MAREK REPKA
Faculty of Electrical Engineering and Information Technology
Institute of Computer Science and Mathematics
Ilkovicova 3, SK-812 19 Bratislava
SLOVAK REPUBLIC
marek.repka@stuba.sk

Abstract: McEliece Public Key Cryptosystem (PKC) is interesting since its resistance against all known attacks, even using quantum cryptanalysis. Unfortunately, Side Channel Attacks (SCAs) are very powerful attacks which even McEliece PKC is vulnerable to. In this work, we present the Leakage Measurement Tool of the McEliece PKC Calculator [1]. The Calculator also provides test vectors (for all important intermediate results). The Calculator implementation is available in [2]. Using the measurement tool, we present a new Template-Timing Analysis of the Patterson’s Algebraic Decoding Algorithm. Essentialy, it is information about the secret Goppa polynomial that leaks from the computation time of some steps in the Patterson’s algebraic decoding algorithm.

Key–Words: Post-Quantum PKI, McEliece PKC, Niederreiter PKC, Patterson’s Algebraic Decoding Algorithm, Binary Irreducible Goppa Codes, Side-Channel Analysis.

December 2, 2014

1 Introduction

Asymmetric cryptography is contemporary based on the integer factorization problem, or the discrete logarithm problem (very often defined over elliptic curves). An alternative to the common asymmetric cryptography is needed as the alternative should be based on a different problem that the common asymmetry, since by breaking those problems, the alternative cryptosystem will be untouched. The common asymmetric cryptography is very vulnerable to the Shor’s quantum algorithm [3], and, therefore, the different problem must be also resistant against quantum cryptanalysis.

Fortunately, there is the McEliece PKC [4] (and also the derived Niederreiter PKC in a particular setup) that might prove resistant to attacks. The McEliece PKC is based on different problems like the usual ones. Those problems are also not affected by the Shor’s algorithm [3]. It is based on the Coset Weights Decision NP-complete Problem, and the Subspace Weights Decision NP-complete Problem [5]. These problems come from coding theory, and, thus, this is a code-based cryptography. For a very fresh survey of cryptography based on error correcting codes consult [6]. Using the McEliece PKC, or others code-based cryptosystems, one can construct all the cryptographic primitives such as encryption, signature, hash functions, pseudo-random generators and so forth.

The original McEliece proposal, which uses random instances of binary irreducible Goppa code with the maximal length, has been unbroken since 1978, even considering quantum cryptanalysis (note the IND-CCA2 conversion importance [7]). There were many attempts to replace the underlying linear error correcting code but all attempts failed except of the new proposal to use the Moderate Density Parity-Check Codes (MDPC) [8]. Since the MDPC proposal is very new, no one of the modifications of the original McEliece proposal is confident in the post-quantum sense today. Nowadays, there are only few, and very limited, implementations of the original proposal. Note also, that many derivatives of the original McEliece PKC have been published. For instance, the Niederreiter PKC [9] is equivalent to the McEliece PKC in terms of security but only if the Binary Irreducible Goppa codes with the maximal length are used. Other derivation is the HyMES [10], but the HyMES differs from the original McEliece PKC proposal in the key-pair and error vector generation, and, thus, also in encryption as well as in decryption. A good implementation of the HyMES with some countermesures implemented and some improvements is the FLEA [11]. Further, a scheme that uses MDPC can be found in [12].

Since the derived schemes are young and, we can say, not sufficiently (quantum) cryptanalysed, no
one of the derived schemes, except for the Niederreiter PKC in the proper setup, is confident as a post-quantum cryptosystem. This is the reason why we decided to implement the original McEliece PKC proposal.

The Calculator implements the original McEliece PKC proposal in order to make the PKC more available. The adjective generic has been achieved using the Number Theory Library (NTL) [13], and the generic CPU Tick Measurement Library [14]. No parameter is fixed in this implementation, and test vectors for all the important intermediate results (for all appropriate \(m\) and \(t\) in limits of hardware and NTL) can be provided for any: encryption, decryption, or key generation. Thanks to the NTL, the Calculator is easy to understand, use, and modify, since the standard NTL functions, input, and output, are used. Therefore, if a key-pair not generated by the Calculator is desired to be used by Calculator, it is not a problem, it must be just formatted accordingly. In this paper, we are focused on the measurement tool for side-channel analysis, which test vectors can be also recorded for. Using this tool, timing leakage can be measured, and using the measured data, it is also possible to simulate power-consumption and electromagnetic-emanation leakages. The CPU Tick Library allows to measure CPU Ticks on different families of processors, and operation systems. There exist several other implementations, like [11, 15, 16, 17, 18], of the original, and derived, schemes on hardware, embedded, and also on a computer platform. However, parameters are fixed, no test vectors are provided, or no tool for the side channel analysis is employed. The Calculator can be used in the PKC implementation optimization, and further McEliece/Niederreiter like PKCs properties investigation, as well as in proper key-pairs generation. The Calculator can be also used in proper parameter choice for a hardware implementation, and the leakage-measurement tool can provide information on side-channel vulnerabilities.

Since we have Post-Quantum PKC in the secure cryptosystem property setup [19], the only possibility how to break this PQ-PKC is via side-channel attacks. Recently, several side-channel attacks have been published [11, 20, 21, 22, 23]. It is possible to attack key generator, decryptor, and also encryptor. We stressed only the Patterson’s algebraic decoding algorithm used in the decryption process. By the tool, secret error vector, secret permutation, and secret Goppa polynomial can be guessed, and the success rate of the guessing can be evaluated.

2 The Template-Timing Analysis

We present Template, or equivalently Profiling, Side-Channel Analysis using Linear Discriminant Analysis as the distinguisher. As the analysis assumption, an adversary must be able to chose HW(\(e\)) before encryption, the adversary must be able to measure computation time during the decryption, and a victim must use very similar implementation as the implementation that was used to create the templates for the attack by the adversary.

Some information about the secret Goppa polynomial that is used in the Patterson’s algorithm as reduction polynomial leaks from the each operation. However, we show that there is one operation that leaks the most significantly in comparison to the other ones. The most significant leakage comes from the square-root computation that is implemented according to Eq. 4 in the Calculator. Such an implementation can be very often seen on memory constraint devices since there is no memory for the look-up table implementation. This square-root is computed in the Step 3 in the Patterson’s algorithm (Alg. 2). From the experimental results we list, it can be seen that there exists some \(g(Z)\) that Hamming Weight of coefficients can be guessed. If, moreover, such a Goppa polynomial has the Hamming Weight low enough, the coefficients can be guessed directly. Otherwise, the guessed Hamming Weight can be misused in other attacks as a side-information, such as for example, in the attack [24].

3 The Side-Channel-Leakage Measurement Tool

The side-channel-leakage measurement tool records Indicators (Tab. 2) measured in order to perform an attack, and information about secret (Tab. 1 and 4) used to compute success rate of an attack. Secret error vector, secret permutation, and secret Goppa polynomial respectively can be guessed using the measured data. Also the power-consumption and electromagnetic-emanation leakages can be simulated. Using this tool, particular keys, and proposed countermeasures can be tested. Thanks to the NTL, source codes are easy to read, and it is possible to replace a measured operation for a designer’s one. Not only computation time is measured by the tool. Also Degrees and Hamming Weights of polynomials processed are recorded. Here are commands for the measurement tool:

```
measure key code pubKeyFile
privKeyFile nTests min_hw_e
max_hw_e measurementFile
```
This measurement tool is employed in the Patterson’s algebraic decoding algorithm (Alg. 2). As we mentioned above, using the tool, secret error vector, secret permutation, and also the secret Goppa polynomial can be possible to guess. Moreover, power consumption and electromagnetic emanation leakages can be simulated using the measured data provided by this tool.

3.1 Decryption

Here we recall the decryption algorithm together with the Patterson’s Algebraic Decoding Algorithm in the way as it is in the Calculator [1].

The Patterson’s Algebraic Decoding Algorithm (Alg. 2) is used in order to correct a code word with $t$ errors in the private code $\Gamma(A, g)$ code. For that purpose, we assume an input binary vector $u = yP^{-1}$ that is a codeword in the private code with exactly $t$ errors. Note, $t$ is the maximum number of errors that can be corrected in a $\Gamma(A, g)$ code, and also that the algorithm in the Alg. 2 is capable to correct.

As the first step of the Alg. 2, the syndrome of the error vector is computed. One can compute the syndrome evaluating the syndrome polynomial, but such an evaluation would be the most time consuming step in the decoding algorithm. On the other hand, such an evaluation is very useful on memory constraint devices. In order to speed up the. Other possibility how to compute the syndrome is to compute the product $uH_{priv}^T$, wherein the $H_{priv}$ is a parity-check matrix of the secret $\Gamma(A, g)$ code, and obtain the syndrome in this way.

The syndrome polynomial $S(e, Z)$ satisfies

$$S(e, Z) = \frac{\sigma'(e, Z)}{\sigma(e, Z)} \mod g(Z).$$

Since the error of a word is being determined, the first derivative of the error-locator polynomial consists only of all the even terms, i.e. the error-locator polynomial can be split into squares and non-squares:

$$\sigma(e, Z) = \alpha^2(e, Z) + Z\beta^2(e, Z),$$

where $\beta^2(e, Z) = \sigma'(e, Z)$. After few modifications, the Key Equation can be obtained as:

$$\beta(e, Z)\sqrt{T(e, Z)} \equiv \alpha(e, Z) \mod g(Z).$$

Therefore, in the Step 3 of the Alg. 2, the square-root modulo $g(Z)$ is computed. Let us denote the term $T^{-1}(e, Z) + Z$ as $T(e, Z)$. The Calculator implementation uses the fact of the perfect square, and thus

$$\sqrt{T(e, Z)} \equiv T^{2^{m-1}}(e, Z) \mod g(Z).$$

Such an approach can be very useful on memory constraint devices, but on the other hand it is very time consuming operation. Another possibility how to compute that square-root is to use precomputed look-up table, which consist of $\tau_i(Z)$ such that $\tau_i^2(Z) \equiv Z^i \mod g(Z)$ for $0 \leq i < t$.

Subsequently, the key equation is solved using the Extended Euclidean Algorithm (EEA) that stops when $\deg \alpha_j(Z) \leq \lceil (t + 1)/2 - 1 \rceil \leq t/2$, where $j$ is the EEA iteration number.

At the time the error-locator polynomial $\sigma(e, Z)$ is computed, roots of the error-locator polynomial shall be found. The Calculator implementation simply evaluate the $\sigma(e, Z)$ over the secret code support $A$. Therefore, Steps 6, 7, 8 are conducted in one loop. This method is also time consuming, and as an alternative, any other factorization method can be employed. Thus, the decoding algorithm yields the error vector $e$ in the private code.
Table 1: Information recorded about secret error vector in order to measure success rate of an attack.

<table>
<thead>
<tr>
<th>Item/Column no</th>
<th>The secret value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HW(e)</td>
</tr>
</tbody>
</table>

Table 2: Indicators measured in order to be able to perform side-channel attacks, and determine where the leakages occur.

<table>
<thead>
<tr>
<th>Item/Column no</th>
<th>Alg. 2 Step</th>
<th>Indicators: avg(.), std(.)</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, ... , 6</td>
<td>1.</td>
<td>computation time, deg, HW</td>
<td>S(Z)</td>
</tr>
<tr>
<td>7, ... , 12</td>
<td>2.</td>
<td>computation time, deg, HW</td>
<td>T(Z)</td>
</tr>
<tr>
<td>13, ... , 18</td>
<td>3.</td>
<td>computation time, deg, HW</td>
<td>τ(Z)</td>
</tr>
<tr>
<td>19, 20</td>
<td>4.</td>
<td>computation time</td>
<td>EEA</td>
</tr>
<tr>
<td>21, ... , 24</td>
<td>4.</td>
<td>deg, HW</td>
<td>α(Z)</td>
</tr>
<tr>
<td>25, ... , 30</td>
<td>5.</td>
<td>computation time, deg, HW</td>
<td>σ(Z)</td>
</tr>
<tr>
<td>31, 32</td>
<td>6. , 7. , 8.</td>
<td>computation time</td>
<td>e constr-</td>
</tr>
</tbody>
</table>
Figure 1: Graph for measurement of computation time in respect to HW(e).

Figure 2: Graph in Fig. 1 without the line for Step 3.

Figure 3: Graph for measurement of HW of polynomial processed in respect to HW(e).

Figure 4: Graph for measurement of deg of polynomial processed in respect to HW(e).
3.4 Measurement Type 2

This second measurement type is designated to measure Indicators (Tab. 2) dependency on secret Goppa polynomial (Tab. 4). Corresponding to each information about Goppa polynomial, Indicators are measured. If moreover dependency on secret permutation is desired to measure, then the flag is_storeKeys must be set to 1. Output file of a measurement of this type is composed as follows.

At the beginning of the file, the measurement setup is presented. The measurement setup is arranged in the Tab. 3. Regarding the measurement setup, afterwards, information about Goppa polynomials (Tab. 4), HW(e) (Tab. 1), and the measured Indicators (Tab. 2) are stored from left to right respectively. Thus, for each Goppa polynomial there is a row, which displays also step-by-step HW(e) and indicators for each $i \in [\min(HW(e)), \max(HW(e))]$ according the measurement setup.

Such as in the measurement type 1, the average values and the standard deviations are computed from nTests encryptions. The measurement file contains nRandKeyPairs measurement rows, each for one randomly generated key pair $(K_{priv}, K_{pub})$.

Since we need to compute a success rate, we need two tuples of $X$ and $Y$, namely, training set $(X_{tr}, Y_{tr})$ and testing set $(X_{ts}, Y_{ts})$, with independent measurements. For training set, we measured the values for nRandKeyPairs = 4000 random key pairs, wherein for each $0 \leq HW(e) \leq 50$, nTests = 3 random messages has been encrypted and decrypted. The testing set was created with the same parameters except for nRandKeyPairs = 14000.

First, it is appropriate to see whether there are some Indicators which are somehow linearly dependent on Hamming Weight of the coefficients. When we compared Pearson’s correlation coefficient with the standard covariance scaled using its range, the covariance has been slightly better, and better success rate has been achieved. Therefore, graphs for the scaled covariance between all the indicators and Goppa polynomial coefficients Hamming Weights are figured. From the figures, it is obvious that the average computation time of the square-root computation (Eq. 4) brings an information. Furthermore, it can be also seen that different HW(e)s reveal different coefficients Hamming Weights. Consequently, we take only those indicators that has $\text{abs}(\text{cov}(X_{tr}, Y_{tr}[i])) > 0.75$ in to the template Template[i] construction using LDA. The new matrix $X$, thus, consists of such values, and, clearly, there are only $\text{avg}(\text{time}(\tau(Z)))$ Indicators for some HW(e)s (as can be seen in the Figs. 6, 7). For each guessed Hamming Weight of coefficient $g_i$, $0 \leq i \leq 49$, a Side Channel Leakage Template, Template[i], is created using the training set $(X_{tr}, Y_{tr})$ according to Eq. 5.

3.5 Basic Examples of Measurement Type 2

Measurements of type 2 are presented on a Template, or equivalently Profiling, Side-Channel Analysis using Linear Discriminant Analysis (LDA) as the distinguisher.

We show that some information about the secret Goppa polynomial, which is used in the Patterson’s algorithm as reduction polynomial, leaks from the square-root computation implemented as in Eq. 4. We show that HW of coefficients of the secret Goppa polynomial is included in this leakage. But we must also note that the knowledge of HW as it is does not bring a significant information for guessing the secret Goppa polynomial regarding the fact that $t$ and $m$ are big in secure cryptosystem setup. On the other hand, there are attacks such as [24] that misuse such a knowledge. What is more important is that there is another information that leaks. It is GCD. This attack is not presented in this work.

3.5.1 Experiment Description and the Leakage Place

Using the indicators measured by the Calculator’s tool, we tried to determine Hamming Weights of many secret Goppa polynomials coefficients. The leakage information is, thus, formed of all the Indicators listed in Tab. 2 for all the possible HW(e). This leakage information is represented as matrix $X$, where the indicators forms columns of the matrix. The Hamming Weights of secret Goppa polynomials coefficients are represented by matrix $Y$, where one column is for one Goppa polynomial coefficient. Both matrices has nRandKeyPairs rows, each for one randomly generated key pair $(K_{priv}, K_{pub})$.

Since we need to compute a success rate, we need two tuples of $X$ and $Y$, namely, training set $(X_{tr}, Y_{tr})$ and testing set $(X_{ts}, Y_{ts})$, with independent measurements. For training set, we measured the values for nRandKeyPairs = 4000 random key pairs, wherein for each $0 \leq HW(e) \leq 50$, nTests = 3 random messages has been encrypted and decrypted. The testing set was created with the same parameters except for nRandKeyPairs = 14000.

First, it is appropriate to see whether there are some Indicators which are somehow linearly dependent on Hamming Weight of the coefficients. When we compared Pearson’s correlation coefficient with the standard covariance scaled using its range, the covariance has been slightly better, and better success rate has been achieved. Therefore, graphs for the scaled covariance between all the indicators and Goppa polynomial coefficients Hamming Weights are figured. From the figures, it is obvious that the average computation time of the square-root computation (Eq. 4) brings an information. Furthermore, it can be also seen that different HW(e)s reveal different coefficients Hamming Weights. Consequently, we take only those indicators that has $\text{abs}(\text{cov}(X_{tr}, Y_{tr}[i])) > 0.75$ in to the template Template[i] construction using LDA. The new matrix $X$, thus, consists of such values, and, clearly, there are only $\text{avg}(\text{time}(\tau(Z)))$ Indicators for some HW(e)s (as can be seen in the Figs. 6, 7). For each guessed Hamming Weight of coefficient $g_i$, $0 \leq i \leq 49$, a Side Channel Leakage Template, Template[i], is created using the training set $(X_{tr}, Y_{tr})$ according to Eq. 5.
Table 5: Measurement setup for Training set \((X_{tr}, Y_{tr})\).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Set value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m)</td>
<td>11</td>
</tr>
<tr>
<td>(t)</td>
<td>50</td>
</tr>
<tr>
<td>(nRandKeyPairs)</td>
<td>4K</td>
</tr>
<tr>
<td>(nTests)</td>
<td>3</td>
</tr>
<tr>
<td>min(HW(e))</td>
<td>0</td>
</tr>
<tr>
<td>max(HW(e))</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 6: Measurement setup for Testing set \((X_{ts}, Y_{ts})\).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Set value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m)</td>
<td>11</td>
</tr>
<tr>
<td>(t)</td>
<td>50</td>
</tr>
<tr>
<td>(nRandKeyPairs)</td>
<td>14K</td>
</tr>
<tr>
<td>(nTests)</td>
<td>3</td>
</tr>
<tr>
<td>min(HW(e))</td>
<td>0</td>
</tr>
<tr>
<td>max(HW(e))</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 5: LDA success rate of guessing HW of the whole goppa polynomial.

Figure 6: Covariance(Indicators, HW\((g_0)\)). Only Indicators with \(\text{COV}/\text{max(abs(COV))} \geq 0.75\) are marked. We can see "avg(time(\(\tau\))).. HW(e)". where \(\tau\) is the square-root. The number for HW(e) = 0 is not printed.

Figure 7: Covariance(Indicators, HW\((g_1)\)). Only Indicators with \(\text{COV}/\text{max(abs(COV))} \geq 0.75\) are marked. We can see "avg(time(\(\tau\))).. HW(e)". where \(\tau\) is the square-root. The number for HW(e) = 0 is not printed.
Figure 8: LDA success rate of guessing HW of coefficients of goppa polynomial. One line is for one coefficient.

\[
\text{Template}[i] = \text{LDA}(\text{Indicators} = X_{tr}, \text{Classes} = Y_{tr}[i]).
\] (5)

As the prior probabilities of classes assignments, the relative classes counts are used in the LDA. In order to determine the success rate, afterwards, the classification is performed using the \(\text{Template}[i]\) over the testing set \((X_{ts}, Y_{ts})\).

\[
\text{OrderedHypotheses}[i] = \text{Classify}(\text{Template} = \text{Template}[i], \text{Indicators} = X_{ts}).
\] (6)

The possible hypotheses are ordered according to the posterior classification probabilities. The result of the relative match with the truth Hamming Weight in \(Y_{ts}[i]\) is depicted in Fig. 8 for guessing HWs of all the coefficients. Also success rate for LDA when guessing HW of the whole goppa polynomial is depicted in Fig. 5.

4 Conclusions

We presented the Side-channel-leakage measurement tool implemented in the Calculator that is an implementation of the original McEliece PKC. We validated the linear dependency of computation time of some steps in decoding algorithm on HW(e). We demonstrated Template-Timing Analysis and showed that there can leakage occur on various places. Therefore, despite the post-quantum cryptography being very resistant to known attacks, problems can arise when such strong cryptographic algorithms, like McEliece PKC or Niederreiter PKC, are implemented in real devices, and post-quantum cryptography is not an exception. Hence, we must implement also such strong algorithms in regards to SCAs threats.

Acknowledgements: Supported in part by NATO’s Public Diplomacy Division in the framework of Science for Peace SPS Project 98452; grant APVV-0586-11; grant VEGA 1/0173/13; National Scholarship Programme of SR – SAIA, n. o.; Laboratoire Hubert Curien UMR CRNS 5516; and the Slovak TEMPEST, a.s. Company.

References:


