Differential Power Analysis of a McEliece Cryptosystem

Cong Chen[1], Thomas Eisenbarth[1], Ingo von Maurich[2], and Rainer Steinwandt[3]

[1] Worcester Polytechnic Institute, Worcester, MA, USA
{cchen3,teisenbarth}@wpi.edu
[2] Ruhr-Universität Bochum, Germany
ingo.vonmaurich@rub.de
[3] Florida Atlantic University, USA
rsteinwa@fau.edu

Abstract. This work presents the first differential power analysis of an implementation of the McEliece cryptosystem. Target of this side-channel attack is a state-of-the-art FPGA implementation of the efficient QC-MDPC McEliece decryption operation as presented at DATE 2014. The presented cryptanalysis succeeds to recover the complete secret key after a few observed decryptions. It consists of a combination of a differential leakage analysis during the syndrome computation followed by an algebraic step that exploits the relation between the public and private key.

Keywords: Differential Power Analysis, McEliece Cryptosystem, QC-MDPC Codes, FPGA

1 Introduction and Motivation

The basic idea of the McEliece public-key encryption scheme can be traced back more than 35 years [19]. Having passed the test of time, today it is considered one of the most promising alternatives to public-key encryption schemes whose underling hardness assumptions are invalidated by known quantum algorithms [23]. A critical point of McEliece-based constructions is the large key size, and to tackle this problem it is tempting to impose additional structure on the code involved. For some proposals in this line of work, including constructions building on Goppa codes, cryptanalytic strategies to exploit the additional structure have been put forward [5,6,7]. Lacking obvious algebraic code structure that can be exploited by an adversary, *quasi-cyclic moderate-density parity-check (QC-MDPC)* codes currently receive considerable attention as an implementation choice [3,9,17,18]. In this paper we take a closer look at a lightweight state-of-the-art FPGA implementation of this scheme as presented in [17].

Our contribution. In this paper we are not concerned with the security of the specific parameters in [17] against the underlying theoretical problem, and instead focus on *side-channel attacks*. Even in a post-quantum world, i. e., when scalable

quantum computers are available, implementation-specific information leakage will remain a serious issue, but so far no differential power analysis (DPA) has been documented on implementations of McEliece. In fact, [10] concluded that a classical DPA attack is not possible for their target implementations. In this paper we demonstrate that DPA can be a realistic threat for a state-of-the-art FPGA implementation of McEliece. Besides showing that significant parts of the private key can be recovered by DPA, we show that knowledge of the public key can be utilized to recover missing key information or to correct remaining errors in hypothesized key bits.

On the conceptual side it deserves to be noted that our cryptanalysis targets the decoding algorithm, and thus is not restricted to a basic McEliece as presented in [17]. If the basic scheme is augmented with a padding to establish stronger provable guarantees, then this does not prevent our side-channel attack as long as the decryption algorithm is applied to the ciphertext directly, possibly followed by some plausibility checks. This type of padding is common in combination with the McEliece cryptosystem [13,22].

Related work. Using QC-MDPC codes in the McEliece cryptosystem was first proposed by [20] and later published with small changes in the parameter set in [21]. These codes have no obvious algebraic structure and still allow small key sizes, which gained high interest in the research community. First implementations of this scheme for AVR microcontrollers and Xilinx FPGAs were proposed in [9]. Their FPGA implementation aimed for a high throughput at the cost of a high resource consumption while their microcontroller implementation for the first time showed that it is possible to implement McEliece without external memory to store the keys. A recent lightweight FPGA implementation showed the full potential of this promising scheme [17]. Occupying less than 230 slices and 4 Block RAMs on Xilinx's smallest Spartan-6 FPGA (XC6SLX4) for a combined encryption/decryption unit, their implementation still provides a reasonable performance of 3.4 ms and 23 ms for en-/decryption, respectively.

Side-channel leakages of McEliece have first been studied in [26]. This work, as well as two follow-up studies focused on analyzing timing behavior of different parts of PC implementations of McEliece [24,25]. Subsequently, [1] improved over prior results, presented countermeasures and pointed out leakages in the preprocessing steps of McEliece encryption. Heyse et al. [10] performed power analysis on software implementations of classic McEliece implementations. Their work relies on simple power analysis (SPA)-based approaches, which usually do not translate well to hardware implementations, due to the increased parallel processing of data and the much smaller side-channel leakage. They also show that side-channel analysis is impeded by the large key sizes of McEliece. In a recent work, AVR/ARM microcontroller implementations of QC-MDPC McEliece were shown to be susceptible to SPA attacks [18]. The found weaknesses rely on secret dependent branches, which allow to recover the encrypted message as well as to recover the secret key.

2 Background

McEliece based on (QC-)MDPC codes is fully described in [21]. To provide the necessary context for our attack, this section gives a brief summary of (QC-) MDPC codes and their instantiation in the McEliece cryptosystem.

2.1 Quasi-Cyclic Moderate-Density Parity-Check Codes

A *binary linear* $[n, k]$ *error-correcting code* C of length n is a k-dimensional vector subspace of $\mathbb{F}_2^n$. We write $r = n - k$ for the co-dimension of C. The code C can be specified by providing a *generator matrix* $G \in \mathbb{F}_2^{k \times n}$, i. e., a matrix whose rows form a basis of C. Alternatively, one can provide a *parity-check matrix* $H \in \mathbb{F}_2^{r \times n}$ which characterizes the linear code as $C = \{c \in \mathbb{F}_2^n | \, cH^T = 0^r\}$. Given a parity-check matrix and a vector $x \in \mathbb{F}_2^n$, we refer to $s = Hx^T \in \mathbb{F}_2^r$ as *syndrome* of x. In particular, a vector from $\mathbb{F}_2^n$ is contained in C if and only if its syndrome is 0^r.

If a code C is closed under cyclic shifts of its codewords by n_0 positions for some integer $n_0 \geq 1$, we refer to C as *quasi-cyclic* (QC). If $n = n_0 \cdot p$ for some integer p, both generator and parity-check matrix can be chosen to be composed of $p \times p$ circulant blocks. This has the advantage that only one row (usually the first) of each circulant block needs to be stored to completely describe the matrices. For a *moderate-density parity-check* (MDPC) code, we choose the weight of each row to have the same density $w = \mathrm{O}(\sqrt{n \log(n)})$. For short, we refer to a binary linear $[n, k]$ error-correcting code defined by a parity-check matrix with constant row weight w and co-dimension r as an (n, r, w)-MDPC code. If such a code is in addition quasi-cyclic with $n = n_0 r$, we speak of an (n, r, w)-QC-MDPC code.

2.2 The QC-MDPC McEliece Public-Key Encryption Scheme

The QC-MDPC McEliece public-key encryption scheme uses t-error correcting (n, r, w)-QC-MDPC codes, i. e., up to t "flipped bits" in any codeword $c \in C$ can be corrected. Specifically, using such a code, key generation, encryption, and decryption operations can be described as follows.

Key-Generation. The secret key is comprised of the first rows $h_0, \ldots, h_{n_0-1} \in \mathbb{F}_2^r$ of the n_0 parity-check matrix blocks $H_0, \ldots, H_{n_0-1}$. These rows are chosen at random and it has to be ensured that their weights—the number of non-zero entries—sum up to w: $\sum_{i=0}^{n_0-1} \mathrm{wt}(h_i) = w$, where wt() denotes the Hamming weight computation function. Iterated cyclic rotation of the h_i yields the parity-check matrix blocks $H_0, \ldots, H_{n_0-1} \in \mathbb{F}_2^{r \times r}$ and thereby the secret parity-check matrix $H = (H_0 | \ldots | H_{n_0-1})$ of an (n, r, w)-QC-MDPC code with $n = n_0 r$. Assuming the last block H_{n_0-1} to be non-singular, the public key is obtained as generator matrix $G = [I_k | Q]$ in standard form, simply concatenating the identity

matrix $I_k \in \mathbb{F}_2^{k \times k}$ with

$$Q = \begin{pmatrix} (H_{n_0-1}^{-1} \cdot H_0)^T \\ (H_{n_0-1}^{-1} \cdot H_1)^T \\ \cdots \\ (H_{n_0-1}^{-1} \cdot H_{n_0-2})^T \end{pmatrix}.$$

Similarly as for the secret key, the public matrix G is determined through its first row. For a textbook version of McEliece the systematic form of G is problematic, but in combination with a conversion to protect against chosen-ciphertext attacks (cf. [13,22]) having the generator matrix G in systematic form is accepted practice.

Encryption. To encrypt a message $m \in \mathbb{F}_2^k$, an error vector $e \in \mathbb{F}_2^n$ of weight $\mathrm{wt}(e) \leq t$ is chosen at random. With this, the ciphertext evaluates to $x = (m \cdot G \oplus e) \in \mathbb{F}_2^n$.

Decryption. To decrypt a ciphertext $x \in \mathbb{F}_2^n$, a t-error correcting (QC-)MDPC decoder Ψ_H is applied to x, recovering $mG \leftarrow \Psi_H(x)$. Since G is in systematic form, the message m can simply be read off from the first k positions of mG.

Parameters. For the implementation investigated in this paper, we used parameters, which in [21] have been considered for an 80-bit security level: $n_0 = 2, n = 9602, r = 4801, w = 90, t = 84$. With these parameters a 4801-bit plaintext block results in a 9602-bit codeword to which $t = 84$ errors are added. The parity-check matrix H has constant row weight $w = 90$ and is obtained as juxtaposition of $n_0 = 2$ circulant blocks. The Q-part of the public generator matrix G consists of a single circulant block.

2.3 Decoding (QC-)MDPC Codes

Several decoders have been proposed to actually decode (QC-)MDPC codes [2,8,9,11,21]. The implementation investigated in this paper employs the decoder from [9], an optimized version of the *bit-flipping decoder* by [8]. The precomputed thresholds are derived from the code parameters as proposed by [8]. To decode a received ciphertext $x \in \mathbb{F}_2^n$, four main steps are involved:

1. Compute the syndrome $s = Hx^T$.
2. Count the number of unsatisfied parity checks for every ciphertext bit.
3. If the number of unsatisfied parity checks for a ciphertext bit exceeds a precomputed threshold, flip the ciphertext bit and update the syndrome.
4. If $s = 0^r$, the codeword was decoded successfully. If $s \neq 0^r$, go to Step 2 or abort after a defined maximum of iterations with a decoding error.

2.4 Target Implementation

The target under investigation is a lightweight implementation of QC-MDPC McEliece for reconfigurable devices by [17]. The resource requirements are 64 slices and 1 block RAM (BRAM) to implement encryption and 159 slices and 3 BRAMs to implement decryption on a Xilinx Spartan-6 XC6SLX4 FPGA. This lightweight implementation is possible mainly for two reasons. First, QC-MDPC codes allow smaller keys compared to (optimized) binary Goppa codes. Second, the implementation stores inputs, outputs and most intermediate values during encryption and decryption in block memories. Since our attack focuses on secret-key recovery, we limit the description of the details of the implementation to the decryption, especially to the part in which the syndrome is computed.

Decryption uses three BRAMs, one BRAM stores the $2 \cdot 4801$-bit secret key, one BRAM stores the $2 \cdot 4801$-bit ciphertext, and one BRAM stores the 4801-bit syndrome. Each BRAM is dual-ported, offers 18/36 kBit, and allows to read-/write two 32-bit values at different addresses in one clock cycle. To compute the syndrome, set bits in the ciphertext select rows of the parity-check matrix blocks that are accumulated. Since only one row of each block is stored in the BRAM, they need to be rotated by one bit to generate the next rows. To generate all rows of H, the rotation is repeated 4801 times.

Rotating the two parts of the secret key is implemented in parallel, which means that the 4801-bit rows of the first and the second part of the parity-check matrix are rotated at the same time. Efficient rotation is realized using the READ_FIRST mode of Xilinx's BRAMs which allows to read the content of a 32-bit memory cell and then to overwrite it with a new value, all within one clock cycle.

The key rotation is implemented as follows: in the first clock cycle, the least significant bit (LSB) is loaded from the last memory cell. The first 32-bit of the row to be rotated are loaded next. In all following clock cycles, the succeeding 32-bit blocks of the row are read and overwritten by the rotated preceding 32-bit block. The LSB of each 32-bit block is delayed by a flip-flop and becomes the most significant bit (MSB) of the following block. An abstraction of this implementation is depicted in Figure 1. In addition to a rotation of the rows, this introduces a rotation of the memory cells. After one 4801-bit rotation, the most significant 32 bits of a parity-check matrix row do not reside in memory cell 0 but in memory cell 1.

The syndrome s is computed by processing the ciphertext x in a bitwise fashion. If the j-th bit is set, i. e., $x_j = 1$, then the j-th row of H is added to the syndrome s. The implementation adds two 32-bit words in parallel: one word of the rotated h_0 and one word of h_1 are processed in each clock cycle.

3 Attack Description

Usually DPA attacks exploit an intermediate state $y = f(x, k)$ that is a function of both a known data item x and a subkey k. The subkey space $\mathcal{K}$ should be

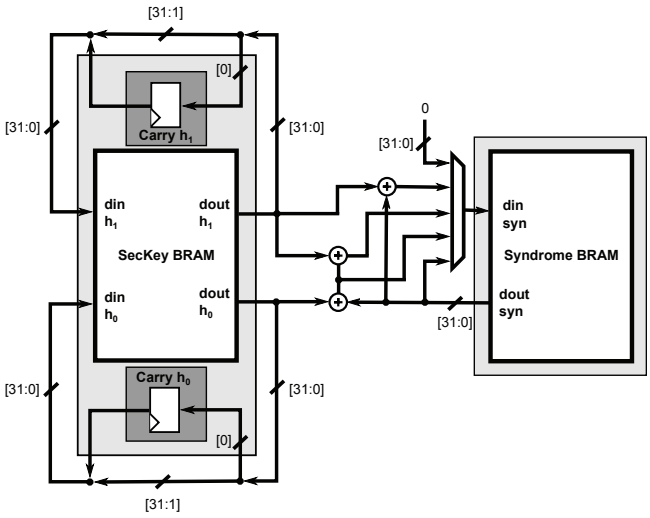


Fig. 1. Abstract block diagram of the syndrome computation circuit including key rotation as implemented in [17].

small enough so that a hypothesis y can be checked for all candidates $k \in \mathcal{K}$. Some works that elaborate on this model are [14,16,27]. McEliece does not offer itself for this approach, as also noted in [10]. One would expect the syndrome s to serve as a potential predictable intermediate state y. However, the bits in the ciphertext x only determine which rows of the parity check matrix H are added to s, where H is the secret key to be recovered. Predicting (parts of) the syndrome s requires an additional key bit hypothesis for each variation of each bit of s, i. e., each bit of s depends on l key bits after l variations, supporting the infeasibility claim of [10]. One of the strengths of QC-MDPC, its small private key size, comes from the fact that secret information is highly redundant: each row of H contains the same information—namely $\langle h_0 \ggg z || h_1 \ggg z \rangle$—only rotated by one bit per row, $z \in \{0, 4800\}$. This redundancy allows for an efficient recovery of key information. More important, it enables a *differential* analysis approach which greatly enhances the visibility of even faint leakages.

We exploit this leakage of the key rotation operation during syndrome computation. Our analysis recovers a static key leakage that is completely independent of the known or chosen ciphertext input x. Since the exploited leakage occurs several times during one syndrome computation, our attack combines these leakage events, as commonly done in horizontal side channel attacks.

3.1 Leakage Behavior of the Target Implementation

The described attack recovers the key during the syndrome computation step of the decryption algorithm. The key for QC-MDPC consists of a single line of the parity check matrix H, namely $h_0||h_1$. As described in Section 2.4, only this line

of H, or one of its rotated versions $\langle h_0 \ggg z || h_1 \ggg z \rangle$, is stored in BRAM. The key has some noteworthy features that influence the derived DPA attacks. First, the private key is of *low weight*: both parts of the secret key h_0 and h_1 are of low Hamming weight such that, $\mathrm{wt}(h_0||h_1) = w$. For the target implementation, $w = 90$ and $\mathrm{wt}(h_i) = 45$, i. e. both h_0 and h_1 have exactly 45 bits set. This means, each key bit $h_{i,j} \in \{0, 1\}$ where $i \in \{0, 1\}$ and $j \in \{0, 4800\}$ is set with probability $\Pr(h_{i,j} = 1) = w/(n_0 r) = 45/4801 \approx .94\%$. This implies *low-weight leakages*: Syndrome and key parts h_i are stored in BRAMs and are processed as 151 32-bit words. The chance of a 32-bit key word to be all-0 is still 74%, about 22% contain a single one bit, leaving the chance of having more than one bit set in a word below 5%.

The critical parts of the target implementation that feature exploitable key leakage are depicted in Figure 1. There are two operations that contribute to the leakage during syndrome computation. One operation is the key rotation, which is always performed. The second operation is the syndrome computation. Our analysis focuses on the key rotation operation, which is independent of the ciphertext input x. The stored key row $\langle h_0 \ggg z || h_1 \ggg z \rangle$ is constantly rotated during the syndrome generation. In fact, it is rotated by a single bit 4801 times, where each rotation takes 151 clock cycles (plus two additional clock cycles for preprocessing and a data read-write delay, resulting in the 153 clock cycles mentioned in [17]). The implementation features a separate register which stores the carry bit during rotations. In each of these clock cycles, one bit $h_{i,j}$—the LSB of the last accessed word—is written to the carry register, causing leakage $\lambda_{\mathrm{carry}}(i, j)$. In the following clock cycle, that bit is overwritten with the LSB of the next word, $h_{i,j+32}$. Assuming a Hamming distance leakage function, this register leaks first

$$\lambda_{\mathrm{carry}}(i, j) = w_1 \cdot \mathrm{wt}(h_{i,j-32} \oplus h_{i,j}), \tag{1}$$

then, in the subsequent clock cycle, leaks $\lambda_{\mathrm{carry}}(i, j+32) = w_1 \cdot \mathrm{wt}(h_{i,j} \oplus h_{i,j+32})$, where $w_1 \in \mathbb{R}$ is an appropriate weight. Assuming that $h_{i,j} = 1$ and further $h_{i,j\pm32} = 0$, $\lambda_{\mathrm{carry}}(i, j)$ gives a clearly distinguishable leakage from the case where $h_{i,j} = 0$. This leakage is the target of the described attack.

In addition to the leakage of the carry register $\lambda_{\mathrm{carry}}(i, j)$ described in Equation (1), there are related leakages happening in the same clock cycles. In fact, when $h_{i,j}$ is written to the carry register, the implementation also reads the word $\langle h_{i,j+1} \dots h_{i,j+32} \rangle$ from the block memory at one address and then stores the word $\langle h_{i,j-32} \dots h_{i,j-1} \rangle$ into the block memory at the same address. Both reading and storing operations will cause leakages at different levels. Assuming a Hamming weight leakage function here, reading data and storing data words leaks as

$$\lambda_{\mathrm{read}}(i, j) = w_2 \cdot \mathrm{wt}(\langle h_{i,j+1} \dots h_{i,j+32} \rangle) \text{ and}$$
$$\lambda_{\mathrm{store}}(i, j) = w_3 \cdot \mathrm{wt}(\langle h_{i,j-32} \dots h_{i,j-1} \rangle),$$

respectively. Here, $w_2 \in \mathbb{R}$ and $w_3 \in \mathbb{R}$ are appropriate weights for the different types of operations. The overall observed leakage is approximated as:

$$\mathcal{L}_i(j) = \lambda_{\text{carry}}(i,j) + \lambda_{\text{read}}(i,j) + \lambda_{\text{store}}(i,j) + \mathcal{N}$$

where $\mathcal{L}_i$ is the overall leakage at the clock cycle where $h_{i,j}$ is written into the carry register and $\mathcal{N}$ is noise, which is assumed to be Gaussian. Please note that the target implementation processes h_0 and h_1 in parallel. This means that the leakage functions $\mathcal{L}_0$ and $\mathcal{L}_1$ for h_0 and h_1 overlap. There are two carry registers (cf. Figure 1), one stores $h_{0,j}$ when the other stores $h_{1,j}$. While these leakages slightly differ, we will not attempt to distinguish them. Instead we recover the combined leakages. That is, we predict the combined leakage $h_\Sigma = h_0 + h_1$, which is still sparse. Note that the addition here is *not* in $\mathbb{F}_2$, i. e., we can distinguish the case where $h_{0,j} = h_{1,j} = 1$ from the case $h_{0,j} = h_{1,j} = 0$, although this case is very rare (and will be ignored in the further description). While the model is not perfect, it describes the observed leakages well enough to base a decent key recovery on it.

As in the classical DPA by Kocher et al. [15], we can now hypothesize the value of each key bit $h_{i,j}$ separately. We further know at which clock cycle the leakage of the carry registers (for the key rotation) occurs. Based on this knowledge, one can build the following attack.

3.2 DPA of Key Rotation

As mentioned above, we do not distinguish $h_{0,j}$ and $h_{1,j}$. Instead, we predict the combined leakage $h_{\Sigma,j} = h_{0,j} + h_{1,j}$. Our key recovery works well for this combined leakage, as explained in Section 5. Note that we know for each key bit $h_{i,j}$ at which clock cycle it is processed (if not, several hypotheses can be checked in parallel by analyzing neighboring clock cycles). In fact, knowing the implementation, it is predictable which key bit $h_{i,j}$ enters the carry register in which clock cycle for the key rotation. We use this information to build a differential power analysis attack. In spite of the independence of the input x we claim the analysis method to be differential leakage analysis, since differential leakage traces can be computed—similar to the approach originally proposed in [15].

Our algorithm identifies all clock cycles where $h_{i,j}$ is written to or overwritten in the carry register in each trace L and extracts that leakage from L. Per processed ciphertext bit, only 150 words are rotated. The additional bit is stored in the carry register. Hence, all rotations together result in a total of $4801 \cdot 150$ carry register overwrites for each h_i. Since there are 4801 bits in h_i, each bit is written to the carry register 150 times. The corresponding clock cycles l are then identified and their corresponding leakage $\mathcal{L}_i(j,l)$ is combined, as done in horizontal SCA. The result is a differential leakage trace Δ_{carry} with only one bin per key bit. In other words, the *difference* between a key bit being zero and a key bit being one can be observed by comparing points of the leakage trace Δ_{carry} horizontally. Since the key is sparse, there are only very few bins that

correspond to a bit $h_{i,j} = 1$, while most bins correspond to a bit $h_{i,j} = 0$. The implicit assumption of all bits leaking the same way is perfectly justified: each bit $h_{i,j}$ takes each column position exactly once, in a specific row. That means due to the rotation, each key bit leaks in every position exactly once, averaging out any position-specific leakages.

In order to detect whether a key bit is set, i. e., $h_{i,j} = 1$, we average over all clock cycles where $h_{i,j}$ is written to the carry register.

$$\begin{aligned}\Delta_{\text{carry}}(j) &= \frac{1}{150}\sum_{l=1}^{150}\left(\mathcal{L}_0(j,l) + \mathcal{L}_1(j,l)\right) \\ &= \text{avg}\left(\lambda_{\text{carry}}(0,j) + \lambda_{\text{read}}(0,j) + \lambda_{\text{store}}(0,j)\right. \\ &\quad \left. + \lambda_{\text{carry}}(1,j) + \lambda_{\text{read}}(1,j) + \lambda_{\text{store}}(1,j)\right)\end{aligned}$$

Since $h_{i,j-32} = 0$ with very high probability, $\Delta_{\text{carry}}(j)$ depends directly on the key bit. Further, $h_{i,j} = 1$ has an even stronger influence on $\Delta_{\text{carry}}(j \pm 32)$, since it leaks through $\lambda_{\text{carry}}(i,j)$ and either $\lambda_{\text{read}}(i,j)$ or $\lambda_{\text{store}}(i,j)$. The dependence of $\Delta_{\text{carry}}(j)$ on neighboring key bits $h_{i,j\pm\delta}$, with $\delta \leq 32$, implies that each set key bit not only results in an increased leakage signal for its own position (i. e., index j), but also in the neighboring positions. Note that due to the differing weights, each set key bit imprints a characteristic shape onto the leakage trace. These shapes can (and actually will) overlap if several key bits in the same region are set. Figure 2 shows the comparison of the simulated leakage trace (red(gray)

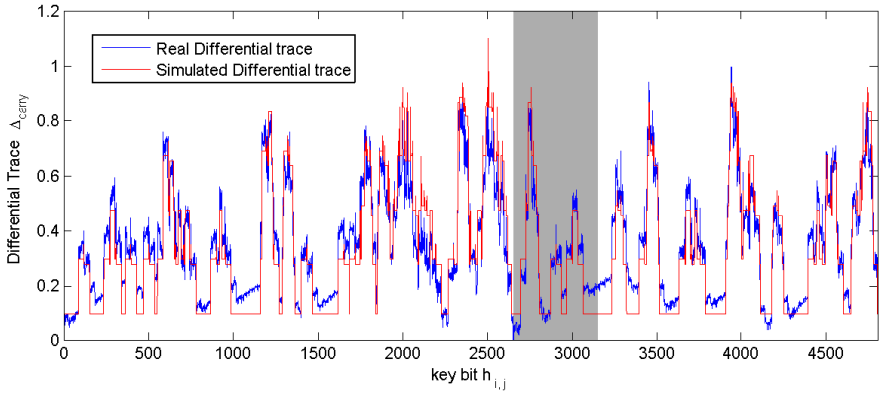


Fig. 2. Differential leakage trace for key rotation. The plot shows the normalized leakage (vertical axis) of both key parts $h_{\Sigma,j} = h_0 + h_1$ over the key bit index (horizontal axis). The red(gray) line is the simulated leakage while the blue/black line is the observed leakage from the target implementation.

line) using the power model and the real leakage trace (blue/black line). The characteristic shape is highlighted in Figure 3, which is a magnification of a single set bit of the key, surrounded by zeroes.

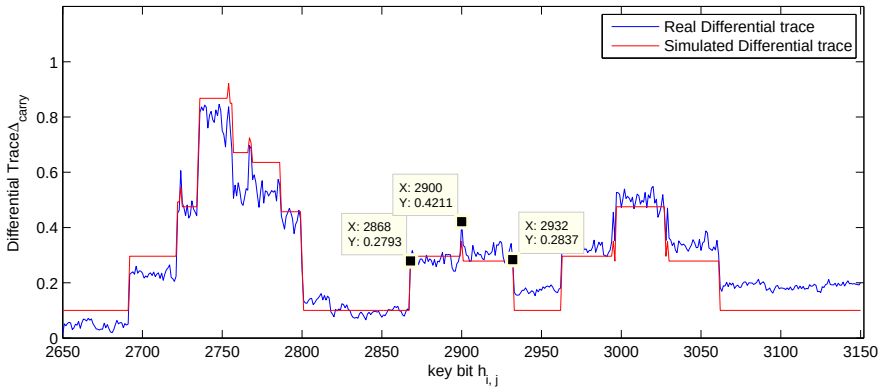


Fig. 3. A magnified version of Figure 2 that highlights the characteristic shape of a single set bit (center) as well as the overlap of two (right) and three (left) "adjacent" set bits.

In summary, the key rotation analysis allows us to detect joint leakages of h_0 and h_1. This is due to the target implementation that processes both in parallel. The key rotation leakage features a characteristic shape with easily detectable bounds. This allows for a precise location of set key bits. Furthermore, the analysis of the key rotation is mostly input-independent, as will be discussed in Section 4. More importantly, each bit features 150 leakage observations per trace L, resulting in a very strong leakage.

3.3 Key Bit Recovery

The key rotation causes leakages which can be analyzed in the presented differential leakage traces where characteristic shapes caused by set key bits can be detected and used to recover the set key bits. In the same way, the traces can be used to detect key bits that are not set. Since the analyzed implementation processes h_0 and h_1 in parallel during the key rotation, resulting in an overlap of the leakages, the differential leakage trace actually recovers the key bits of $h_\Sigma = h_0 + h_1$.

In order to recover key bits, the characteristic shapes need to be detected. We propose a generic shape detection algorithm that works as follows:

1. **Shape Definition** From the differential leakage trace, one singular characteristic shape can be identified and used as a template for set bits. The template is used to generate a shape threshold as shown in Figure 3. The threshold is defined by the value of features in this shape such as edges, slopes and pulses.
2. **Shape Detection** For each key bit in the differential leakage trace, we check if this key bit together with the neighboring key bits can form a characteristic shape. This is done by checking if there are features that are beyond the

threshold. If more than two features exist, it is highly probable that this key bit is set. If no feature exists, then it is highly probable that this key bit is 0. Otherwise, we mark this key bit as an undetermined bit.

Note that the shapes will overlap if two set key bits are close to each other. Furthermore, the leakage traces are noisy, hence we can only recover parts of the key bits, leaving the other key bits as undetermined. By choosing the thresholds for shape detection carefully, the number of detected bits can be maximized while keeping the number of false positive errors as low as needed.

4 Measurement Setup and Results

We ported the implementation of [17] to a Xilinx Virtex-5 LX50 FPGA which is mounted on a Sasebo-GII side-channel attack evaluation board[1]. The implementation is clocked at 3 MHz by default. Measurements were performed using a Tektronix DPO 5104 oscilloscope at a sampling rate of 100 MS/s. Since our attack focuses on the syndrome computation, only the syndrome computation was recorded. The syndrome computation takes 245 ms, resulting in long traces. For the ease of analysis, a peak extraction was performed. In each clock cycle only the point of maximum power consumption is retained. The peak extraction prevents potential alignment issues and makes data handling much faster.

4.1 DPA Results of the Key Rotation

Since the key rotation is independent of the ciphertext, the choice of the ciphertext could be arbitrary. However, key rotation and syndrome computation run in parallel, leading to a mixed leakage. To determine the influence of the syndrome computation, two different ciphertext scenarios are studied. One is the all-0 ciphertext to minimize the influence of the syndrome computation. In this scenario the syndrome remains all-0 throughout the entire computation. The other scenario assumes random ciphertexts for each decryption, where each bit in x is set with a 50% probability. For each scenario we took 256 measurements.

Next, we averaged over all considered traces in both scenarios. From the resulting average trace, $4801 \cdot 150$ peaks are extracted and used to construct the differential leakage traces Δ_{carry} as explained in Section 3.2. Note that averaging explicitly before the computation of Δ_{carry} or implicitly during the computation of Δ_{carry} does not influence the result. Figure 4 shows the differential leakage traces for the key rotation, showing the key bit position (horizontal axis) vs. the bit leakage (vertical axis) for all key bits. The blue (black) line indicates the result for the all-0 ciphertext scenario while the green (gray) line indicates the results for the random ciphertext. The latter one is slightly noisier, but nevertheless provides a well-exploitable leakage for a low number of observations. Figure 3 shows magnifications of the differential leakage trace to highlight the

[1] The VHDL code of the QC-MDPC McEliece implementation of [17] is available at `http://www.sha.rub.de/research/projects/code/`.

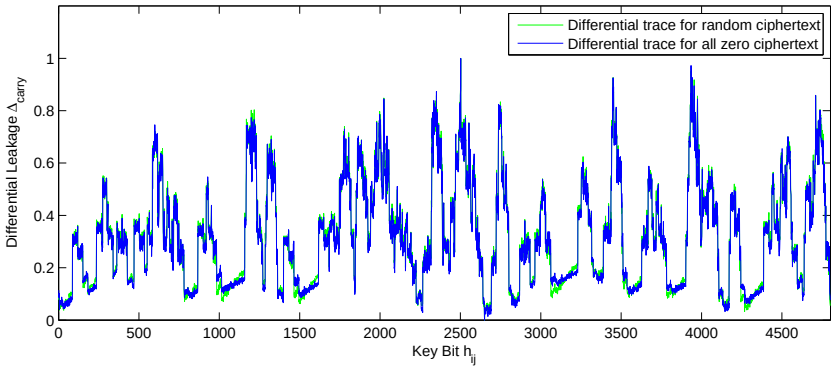


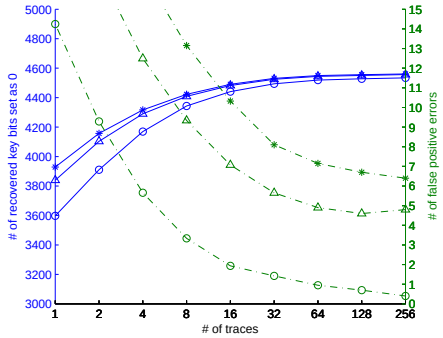
Fig. 4. Normalized differential leakage trace Δ_{carry} for the key rotation for the bits of $h_{\Sigma,j} = h_0 + h_1$. Whether the ciphertext is known (green(gray) line) or all-0 (blue(black) line) has only marginal influence on the observed leakage.

characteristic shapes, particularly the one generated by setting the key bit $h_{i,2900}$ as 1 and the neighboring key bits as 0.

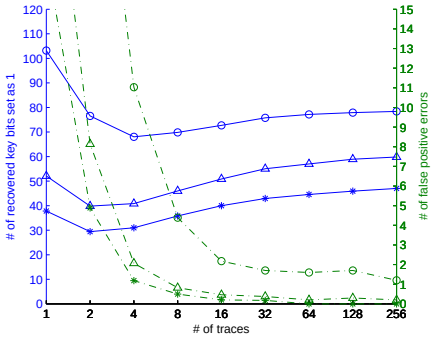
The other shapes in Figure 3 result from the overlapping of characteristic shapes that occur when set key bits of h are close to each other. We noticed that set key bits for h_0 result in a slightly different shape than those of h_1. Since this difference cannot be distinguished as easily, we did not further try to exploit this information.

Key Extraction. To extract keys from Δ_{carry}, we used the algorithm described in Sec. 3.3. The first step is to define the characteristic shape. Distinguishable features such as the rising edge, the pulse in the center and the falling edge are clearly visible in Figure 3 and are used to detect the shape. These features are quantified using a threshold vector. Then, for each key bit $h_{i,j}$ in Δ_{carry}, we check if there is a pulse at $h_{i,j}$, a rising edge at $h_{i,j-32}$ and a falling edge at $h_{i,j+32}$. If more than one feature exists for $h_{i,j}$, we take $h_{i,j}$ as 1. If no feature exists, $h_{i,j}$ is taken as 0. If only one feature exists, $h_{i,j}$ is left as undetermined key bit. Depending on the number of traces used for generating Δ_{carry}, it can be noisy and there will be false positive errors in recovered key bits. Errors can also be introduced by unfavorable overlapping of shapes.

Figure 5 shows how the chosen threshold affects the key recovery. Three different thresholds are used. The first one ($\circ$) is exactly the value extracted from the characteristic shape in Δ_{carry}. The other two ($\triangle$ and then $*$) are increased based on the first one. In Figure 5.1, as the number of traces used to generate the differential leakage trace increases, the number of recovered 0 key bits increases and the number of false positive errors decreases for all three thresholds. However, the less aggressive the threshold is, the lower is the number of false positive errors. In contrast, Figure 5.2 shows that with the least aggressive threshold ($\circ$), more key bits of 1 can be recovered with a few more false positive errors. Hence, to recover more key bits of 0 with least false positive errors, the less aggressive threshold should be used. In contrast, to recover key bits of 1 with



5.1: Recovered 0 bits vs. false positives



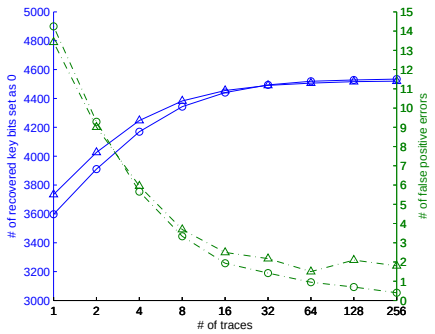
5.2: Recovered 1 bits vs. false positives

Fig. 5. Key bit recovery rates for a range of detection thresholds for recovering 0 key bits (left) and 1 key bits (right). Solid line indicates the number of recovered bits (out of 90 ones and 4711 zeroes, scale on left), the dashed line indicates the number of false positives (scale on right). Markers ∘, then △, and then ∗ indicate the increasing values for the threshold.

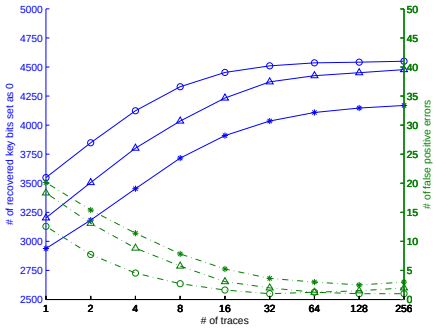
least false positive errors, the more aggressive threshold should be used. Note that we repeated our experiments for five different randomly generated keys to ensure the result is not key dependent. The figures show the average result for those experiments.

Figure 6.1 shows a comparison of the number of recovered key bits and false positive errors between the all-0 ciphertext and random ciphertext. As the number of traces used to generate the differential leakage trace increases, the number of recovered key bits of 0 increases and the number of false positive errors decreases for both cases. However, with the all-0 ciphertext, there are less positive errors. In conclusion, the all-0 ciphertext is more advantageous to the DPA of key rotation. Hence, we use the traces with the all-0 ciphertext in the other experiments.

Modern electronic devices run faster than 3 MHz which is the default clock rate for the SASEBO board and widely used in power analysis experiments. In order to validate our attack on faster platforms, the performance of the attack was measured for the same design clocked at 8 MHz and 16 MHz. The sampling rate was accordingly increased to to 200 MS/s and 250 MS/s, respectively. For each case, 256 traces were obtained using the all-0 ciphertext, followed by peak extraction. Figure 6.2 shows the degradation of the leakage over the increasing clock rate by comparing the number of recovered 0 key bits and false positive errors. In all three cases, the number of recovered 0 key bits increases and the number of false positive errors decreases, as the number of analyzed traces increases. However, the lower the clock rate is, the better the key bits extraction works. With a 3 MHz clock rate (∘), almost 4500 of the 0 key bits can be recovered with about 1 false positive error when using all 256 traces while 4000 of the



6.1: Random vs. all-0 input



6.2: Varying clock rates

Fig. 6. Key bit recovery rates for recovering 0 key bits. Solid line indicates the number of recovered bits (out of 4711 zeroes, scale on left), the dashed line indicates the number of false positives (scale on right). The left figure compares *known* random (○) vs. *chosen* all-0 (△) ciphertext inputs. The right figure compares the experiments for varying clock rates: ○ for 3 MHz, △ for 8 MHz, and ∗ for 16 MHz.

0 bits are recovered with about 3 false positive errors at a clock rate of 16 MHz (∗).

Overall, it can be seen that with as little as 10 measurements, more than half the key bits can be recovered with a remaining number of errors that is small enough to allow for efficient error correction. With 100 measurements and a careful choice of thresholds, the determined bits are entirely error-free at lower clock rates. This strong leakage is partially due to the fact that 150 leakages are extracted from each measurement, strongly amplifying the amount of leakage gained from each individual trace.

5 Full Key Recovery

Next we analyze how to recover the full key of QC-MDPC McEliece if the adversary has knowledge of several 1 bits of the key as well as several 0 bits of the key, possibly with few errors. We show that the structure of the key can be used to recover the remaining uncertain bits efficiently, or to detect remaining errors.

5.1 Exploiting a Connection between Secret Key and Public Key

As described in Section 2.2, the secret key consists of two related parts, h_0 and h_1. Due to the relation between the secret h_0, h_1 and the public matrix Q, we can express h_0 as:

$$h_0 = h_1 \cdot Q^T \tag{2}$$

Likewise, given h_0, one can compute h_1, since Q is invertible. This means that once the first half of the secret key is recovered, the second half can be computed

using the public key. More interestingly, this relationship can be used for *error detection* for each h_i independently: since Q is of high weight (each bit has approximately a 50% chance of being 1), even a single bit error in h_i^* will result in a high weight of a consequently derived $h_{\bar{i}}^*$, i. e., $\mathrm{wt}(h_{\bar{i}}^*) \approx r/2$. A correct h_i, however, will result in an $h_{\bar{i}}$ of low weight, in our case $\mathrm{wt}(h_{\bar{i}}) = 45$. We are currently not aware how slightly faulty or noisy information of h_0 and h_1 can be combined more efficiently without a trial and error approach using the abovementioned relationship.

If the adversary observes a combined leakage of h_0 and h_1 as described above, this is not a problem, since knowledge of $h_0 \oplus h_1$ can also enable key recovery. Adding h_1 on both sides of Equation (2) we obtain

$$h_0 \oplus h_1 = h_1 \cdot (Q^T \oplus I_{4801}). \tag{3}$$

If side-channel leakage allows us to obtain the combined leakage $h_0 \oplus h_1$ and the rank of $Q^T \oplus I_{4801}$ is high, we can solve this linear system of equations for h_1 with a computer algebra system like Magma [4]—and then derive h_0 from Equation (2). In our experiments, the rank observed for $Q^T \oplus I_{4801}$ was 4800, resulting in two candidate solutions with only one of them having the correct Hamming weight. So in cases where all ones can be correctly identified, Equations (2) and (3) enable a practical key recovery.

Due to noise and leakage overlapping, there are probably false positive errors in the recovered bits and hence error correction would be essential to correct positions that are slightly off. Guessing error positions becomes infeasible quickly, even with small improvements over an exhaustive search of $\binom{4801}{l}$ possibilities for l errors. We did not try to devise elaborate error-correction strategies, as a different attack strategy which relies on exploiting detected zeroes turned out to be quite effective. We explain this strategy next.

5.2 Efficient Key Recovery from Partial Information

After having identified several 1 bits and 0 bits of the secret key correctly, we aim at an efficient way to recover remaining unknown or uncertain key bits. For this, we define B_0, B_1 and B_u as index sets indicating the locations of definite zeroes, definite ones and positions of undetermined bits in $h_0 \oplus h_1$ such that

$$B_0 \,\dot\cup\, B_1 \,\dot\cup\, B_u = \{0, 1, \ldots, 4800\}\,. \tag{4}$$

Positions in B_0 indicate that both h_0 and h_1 are zero in that position, while positions in B_1 will mean a one in either h_0 or h_1.[2] Hence, the uncertain positions for h_1 are $B_u^1 = B_1 \,\dot\cup\, B_u$, and with Iverson's convention [12] we can summarize our knowledge of $h_0 \oplus h_1$ and h_1 as $h_0 \oplus h_1 = \langle 1 \cdot [i \in B_1] + u \cdot [i \in B_u] \rangle_{0 \le i \le 4800}$ and $h_1 = \left\langle u \cdot [i \in B_u^1] \right\rangle_{0 \le i \le 4800}$, where u indicates unknown bits ("erasures").

[2] The (rare) case of h_0 and h_1 having a one in the same position is not considered here, as this situation is quite apparent from the side-channel leakage.

So Equation (3) yields

$$\langle 1 \cdot [i \in B_1] + u \cdot [i \in B_u] \rangle_{0 \le i \le 4800} = \left\langle u \cdot [i \in B_u^1] \right\rangle_{0 \le i \le 4800} \cdot (Q^T \oplus I_{4801}).$$

As the indices in B_0 indicate definite zeroes in $h_0 \oplus h_1$ and h_1, the corresponding *rows* in the matrix $Q^T \oplus I_{4801}$ will always be multiplied with a zero coefficient. We remove these $|B_0|$ rows and the corresponding known 0-entries in h_1, obtaining an updated equation system

$$\langle 1 \cdot [i \in B_1] + u \cdot [i \in B_u] \rangle_{0 \le i \le 4800} = \left\langle u \cdot [i \in B_u^1] \right\rangle_{i \notin B_0} \cdot Q'. \tag{5}$$

with a (smaller) matrix $Q' \in \mathbb{F}_2^{(4801-|B_0|)\times 4801}$. There are $4801 - |B_0| - |B_1|$ unknown bits on the left- and $4801 - |B_0|$ unknown bits on the right-hand side of Equation (5). As we are only interested in finding h_1, we can try to eliminate unknown values in $h_0 \oplus h_1$ by dropping *columns* from Q'. One may hope that $|B_u|$ columns can be eliminated without Q' dropping in rank, so that we end up with a linear system of equations

$$\langle 1 \cdot [i \in B_1] \rangle_{i \notin B_u} = \left\langle u \cdot [i \in B_u^1] \right\rangle_{i \notin B_0} \cdot Q'' \tag{6}$$

in $4801 - |B_0|$ unknowns and a matrix $Q'' \in \mathbb{F}_2^{(4801-|B_0|)\times(4801-|B_u|)}$. If $|B_u| \le |B_0|$ one may hope that this linear system of equations can be solved and yields a unique candidate for h_1.

To check the practical feasibility of this approach, we ran several experiments in Magma [4], solving the equation system given in (6) for several different vectors B_0 and B_1. We were particularly interested in the situation where knowledge of 1-positions in $h_0 \oplus h_1$ is ignored (i. e., $B_1 = \emptyset$), because in our measurements the 0-detection was more reliable. With $B_1 = \emptyset$, the resulting system of equations is homogeneous and thus in addition to h_1 also has the trivial solution. From Equation (4) we see that the condition $|B_u| \le |B_0|$ now implies that $|B_0| \ge \lceil 4801/2 \rceil$. Staying above this threshold, in our experiments we obtained no more than 8 candidates for h_1, and the weight condition identified the correct secret key uniquely.

For $|B_0| < 2400$, the kernel of the matrix Q'' in Equation (6) gets larger quickly and we obtain additional candidates for h_1, but finding the correct h_1 may still be feasible by looking at the Hamming weight of the candidates as long as the number of candidates is not overwhelming. The results in Section 4 show that for the target implementation the attacker can expect to recover more information from the side-channel than necessary for recovering the secret key. Having $|B_0|$ comfortably above the threshold of 2400, a few false positives in B_0 can be dealt with efficiently: Instead of using all of these bit positions, one can select subsets of size 2401 at random. Assuming a hypergeometric distribution, with f false positive errors among the $|B_0|$ indices, the probability of guessing 2401 error-free positions is $\binom{|B_0|-f}{2401} / \binom{|B_0|}{2401}$. E. g., with $|B_0| = 3281$ and $f = 4$, this probability is still $\approx 2^{-7.6}$. In summary, as long as more than half the bits of the key can be recovered with a low error rate, the remaining key bits

can be determined using the above-described algebraic methods. Knowledge of additional bits of $h_0 \oplus h_1$ facilitates the handling of possibly remaining errors. Not being able to recover more than half the number of key bits can make the search infeasible, although—due to the highly biased key—guessing a few additional zeroes may still be an option.

6 Preventing the Attack

The described attack is somewhat specific to the implementation choices of the target, but can be adjusted to other implementation parameters as well. For example, an implementation that does not process h_0 and h_1 in parallel would simplify the attack and amplify the leakage. Implementations that use a different word size (the targeted implementation processes 32-bit words due to the BRAM structure of the FPGAs) will influence the described attack as well. The smaller the word size, the more leakages per target bit, most likely facilitating the attack further. However, a massively parallelized implementation such as the one described in [9] could impede the described attack, since all bits would always be leaking in parallel. One might still be able to exploit resource-specific leakages, e. g., leakage from a carry register. Furthermore, such an implementation is very resource-consuming and might not find widespread use.

A more reliable way to prevent this attack is provided by side-channel countermeasures. A good overview of standard DPA countermeasures is available in [16]. Countermeasures are typically classified as *masking* or *hiding* countermeasures. Both classes can be applied to an implementation of (QC-)MDPC McEliece and, if done correctly, should prevent the above-mentioned attack.

7 Conclusion

This work presents the first successful differential power analysis of a state-of-the-art McEliece implementation based on quasi-cyclic MDPC codes. The analysis is not affected by a potentially present padding as commonly used to achieve CCA security. The analysis exploits the leakages of a key rotation operation which occurs during the syndrome computation step of the decryption and recovers a combined leakage of h_0 and h_1. The leakage model provides precise and strong leakage. The resulting attack is independent of the ciphertext and succeeds with tens of traces. A significant part of the key recovery stems from the relation between the private key and public key, which can be exploited to ease key recovery. In fact, recovering only half the bits of the (highly biased) secret key with a low error rate is sufficient for full key recovery.

Acknowledgments. This work is supported by the National Science Foundation under grant CNS-1261399 and grant CNS-1314770. IvM is supported by the European Union H2020 PQCrypto project (grant no. 645622) and the German Research Foundation (DFG). RS is supported by NATO's Public Diplomacy Division in the framework of "Science for Peace", Project MD.SFPP 984520.

References

1. Avanzi, R., Hoerder, S., Page, D., Tunstall, M.: Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems. Journal of Cryptographic Engineering 1(4), 271–281 (2011)
2. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.: On the Inherent Intractability of Certain Coding Problems (Corresp.). IEEE Transactions on Information Theory 24(3), 384–386 (May 1978)
3. Biasi, F.P., Barreto, P.S.L.M., Misoczki, R., Ruggiero, W.V.: Scaling efficient code-based cryptosystems for embedded platforms. J. Cryptographic Engineering 4(2), 123–134 (2014), `http://dx.doi.org/10.1007/s13389-014-0070-1`
4. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. Journal of Symbolic Computation 24, 235–265 (1997)
5. Faugère, J.C., Otmani, A., Perret, L., de Portzamparc, F., Tillich, J.P.: Folding Alternant and Goppa Codes with Non-Trivial Automorphism Groups. Cryptology ePrint Archive: Report 2014/353 (May 2014), available at `http://eprint.iacr.org/2014/353`
6. Faugère, J.C., Otmani, A., Perret, L., de Portzamparc, F., Tillich, J.P.: Structural Cryptanalysis of McEliece Schemes with Compact Keys. Cryptology ePrint Archive: Report 2014/210 (March 2014), available at `http://eprint.iacr.org/2014/210`
7. Faugère, J.C., Otmani, A., Perret, L., Tillich, J.P.: Algebraic Cryptanalysis of McEliece Variants with Compact Keys. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010. Leture Notes in Computer Science, vol. 6110, pp. 279–298. International Association for Cryptologic Research, Springer, Berlin Heidelberg (2010)
8. Gallager, R.: Low-density Parity-check Codes. Information Theory, IRE Transactions on 8(1), 21–28 (1962)
9. Heyse, S., von Maurich, I., Güneysu, T.: Smaller Keys for Code-Based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices. In: Bertoni, G., Coron, J.S. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2013. Lecture Notes in Computer Science, vol. 8086, pp. 273–292. Springer, Berlin Heidelberg (2013)
10. Heyse, S., Moradi, A., Paar, C.: Practical Power Analysis Attacks on Software Implementations of McEliece. In: Sendrier, N. (ed.) Post-Quantum Cryptography – PQCrypto 2010. Lecture Notes in Computer Science, vol. 6061, pp. 108–125. Springer, Berlin Heidelberg (2010)
11. Huffman, W.C., Pless, V.: Fundamentals of Error-Correcting Codes. Cambridge University Press, United Kingdom (2010)
12. Knuth, D.E.: Two Notes on Notation. The American Mathematical Monthly 99(5), 403–422 (May 1992)
13. Kobara, K., Imai, H.: Semantically Secure McEliece Public-Key Cryptosystems – Conversions for McEliece PKC–. In: Kim, K. (ed.) Practice and Theory in Public Key Cryptosystems – PKC '01. Lecture Notes in Computer Science, vol. 1992, pp. 19–35. Springer, Berlin Heidelberg (2001)
14. Kocher, P., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. Journal of Cryptographic Engineering 1(1), 5–27 (2011)
15. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) Advances in Cryptology – CRYPTO'99. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer, Berlin Heidelberg (1999)

16. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smartcards. Springer, US (2007)
17. von Maurich, I., Güneysu, T.: Lightweight Code-based Cryptography: QC-MDPC McEliece Encryption on Reconfigurable Devices. In: Design, Automation and Test in Europe – DATE 2014. pp. 1–6. IEEE (2014)
18. von Maurich, I., Güneysu, T.: Towards Side-Channel Resistant Implementations of QC-MDPC McEliece Encryption on Constrained Devices. In: Mosca, M. (ed.) Post-Quantum Cryptography, Lecture Notes in Computer Science, vol. 8772, pp. 266–282. Springer (2014), http://dx.doi.org/10.1007/978-3-319-11659-4_16
19. McEliece, R.J.: A Public-Key Cryptosystem Based On Algebraic Coding Theory. Deep Space Network Progress Report 44, 114–116 (Jan 1978)
20. Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.L.M.: MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes. Cryptology ePrint Archive, Report 2012/409 (2012), http://eprint.iacr.org/2012/409
21. Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.L.M.: MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes. In: Proceedings of the 2013 IEEE International Symposium on Information Theory (ISIT). pp. 2069–2073. IEEE (2013)
22. Nojima, R., Imai, H., Kobara, K., Morozov, K.: Semantic security for the McEliece cryptosystem without random oracles. Designs, Codes and Cryptography 49(1–3), 289–305 (December 2008)
23. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms On a Quantum Computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
24. Shoufan, A., Strenzke, F., Molter, H., Stöttinger, M.: A Timing Attack against Patterson Algorithm in the McEliece PKC. In: Lee, D., Hong, S. (eds.) Information, Security and Cryptology – ICISC 2009, Lecture Notes in Computer Science, vol. 5984, pp. 161–175. Springer, Berlin Heidelberg (2010)
25. Strenzke, F.: A Timing Attack against the Secret Permutation in the McEliece PKC. In: Sendrier, N. (ed.) Post-Quantum Cryptography – PQCrypto 2010. Lecture Notes in Computer Science, vol. 6061, pp. 95–107. Springer, Berlin Heidelberg (2010)
26. Strenzke, F., Tews, E., Molter, H.G., Overbeck, R., Shoufan, A.: Side Channels in the McEliece PKC. In: Buchmann, J., Ding, J. (eds.) Post-Quantum Cryptography – PQCrypto 2008. Lecture Notes in Computer Science, vol. 5299, pp. 216–229. Springer, Berlin Heidelberg (2008)
27. Whitnall, C., Oswald, E., Standaert, F.X.: The Myth of Generic DPA...and the Magic of Learning. In: Benaloh, J. (ed.) Topics in Cryptology – CT-RSA 2014. Lecture Notes in Computer Science, vol. 8366, pp. 183–205. Springer, International Publishing (2014)