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, Hardware Implementation

1 Introduction and Motivation

The basic idea of the McEliece public-key encryption scheme can be traced back more than 35 years [McEliece, 1978]. 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 [Shor, 1997]. 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 [Faugère et al., 2010, 2014a,b].

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, and in this paper we take a closer look at a state-of-the-art implementation of such a scheme, presented in [von Maurich and Güneysu, 2014].

Our contribution In this paper we are not concerned with the security of the specific parameters in [von Maurich and Güneysu, 2014] 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, [Heyse et al., 2010] 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 [von Maurich and Güneysu, 2014]. 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 McEliece public-key encryption [Kobara and Imai, 2001, Nojima et al., 2008].

Related work Using QC-MDPC codes in the McEliece cryptosystem was first proposed by [Misoczki et al., 2012] and later published with small changes in the parameter set in [Misoczki et al., 2013]. These codes have no obvious algebraic code 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 by [Heyse et al., 2013]. 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 lightweight FPGA implementation by [von Maurich and Güneysu, 2014] showed the full potential of this promising scheme. 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 [Strenzke et al., 2008]. This work, as well as two follow-up studies have focused on analyzing timing behavior of different parts of contemporary PC implementations of McEliece [Strenzke, 2010, Shoufan et al., 2010]. Subsequently, [Avanzi et al., 2011] improved over prior results, presented countermeasures and pointed out leakages in the preprocessing steps of McEliece encryption.

[Heyse et al., 2010] 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.

2 Background

McEliece based on (QC-)MDPC codes is fully described in [Misoczki et al., 2013]. To provide the necessary context for our attack, the remainder of this section gives a brief summary of (QC-)MDPC codes and their proposed use to instantiate 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

For implementing the McEliece encryption scheme, t-error correcting (n, r, w)-QC-MDPC codes are used, 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 must be ensured that their weights—the number of non-zero entries—sum up to w:

$$\sum_{i=0}^{n_0-1} \mathrm{wt}(h_i) = w.$$

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 fully 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. [Kobara and Imai, 2001, Nojima et al., 2008]) 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 [Misoczki et al., 2013] 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 [Berlekamp et al., 1978, Gallager, 1962, Heyse et al., 2013, Huffman and Pless, 2010, Misoczki et al., 2013]. The implementation investigated in this paper employs the decoder from [Heyse et al., 2013], an optimized version of the *bit-flipping decoder* by [Gallager, 1962]. 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.

The precomputed thresholds are derived from the code parameters as proposed by [Gallager, 1962].

2.4 Target Implementation

The target under investigation is a lightweight implementation of QC-MDPC McEliece for reconfigurable devices by [von Maurich and Güneysu, 2014]. 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 XC6SLX4 which is the smallest device available in Xilinx's Spartan-6 family. Encrypting a plaintext takes 3.4 ms and decrypting a ciphertext takes 23 ms.

This lightweight implementation is possible mainly for two reasons. First, QC-MDPC codes allow smaller keys compared to (optimized) 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 that are accumulated. Since only one row of the parity-check matrix is stored in the BRAM, it needs to be rotated by one bit to generate the next row. To generate all rows of H this 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.

Rotation is implemented as follows: in the first clock cycle, the least significant bit 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 least significant bit of each 32-bit block is delayed by a flip-flop and becomes the most significant bit of the following block. An abstraction of this implementation is depicted in Figure 1. In addition to a rotation of the row, this introduces a rotation of the memory cells, so after one 4801-bit rotation, the most significant 32 bits of the parity-check matrix row do not reside in memory cell 0 but in memory cell 1.

The syndrome computation is done in parallel to the key rotation. Ciphertext x is split into two parts that are processed bit-by-bit in parallel. If a bit in one or both parts is set, then the current row(s) h_0/h_1 of the secret key are added to the syndrome, as shown in Figure 1.

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 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 [Mangard et al., 2007, Kocher et al., 2011, Whitnall et al., 2014]. McEliece does not offer itself for this approach, as also noted in [Heyse et al., 2010]. 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 [Heyse et al., 2010]. 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 $\langle h_0 \lll i || h_1 \lll i \rangle$—only rotated by one bit per row. This redundancy allows for an efficient recovery of key information. More importantly, it enables a *differential* analysis approach which greatly enhances the visibility of even faint leakages.

We exploit two different types of leakage, both occurring during syndrome computation. The first analysis recovers key leakage from the syndrome computation itself, as suggested above. The second analysis recovers a static key leakage that is completely independent of the known or chosen ciphertext input x. In spite of

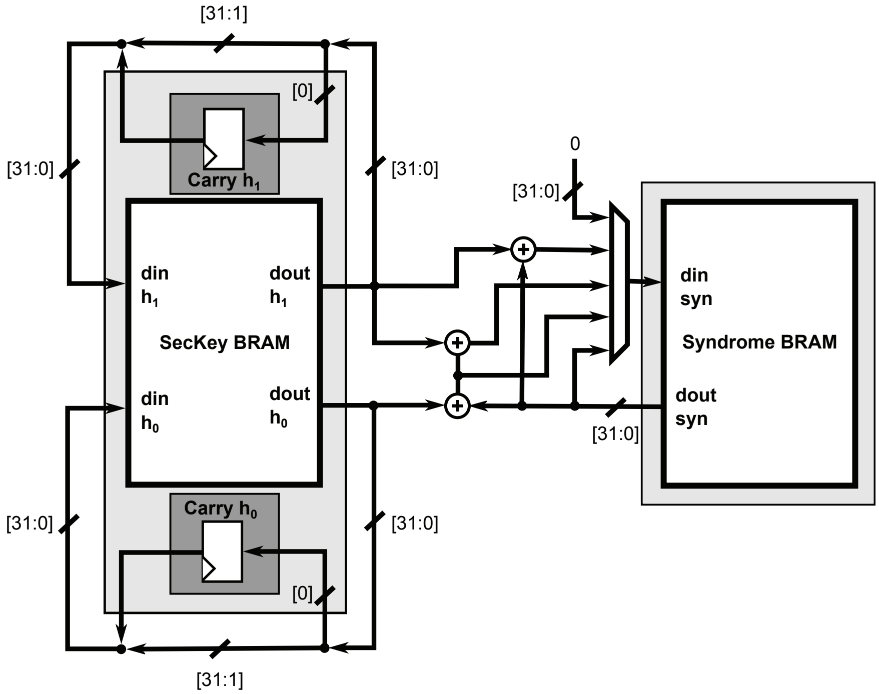


Fig. 1. Abstract block diagram of the syndrome computation circuit including key rotation.

the independence of x we claim both analysis methods to be differential leakage analyses, since in both cases differential leakage traces can be computed—similar to the approach originally proposed by [Kocher et al., 1999].

3.1 Leakage Behavior of the Target Implementation

The described attacks recover 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 $\langle h_0 \lll i || h_1 \lll i \rangle$, 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 $h_{i,j} \in \{0, 1\}$ is set with probability

$$\Pr(h_{i,j} = 1) = \frac{w}{n_0 r} = \frac{45}{4801} \approx .94\%.$$

This implies *low-weight leakages*: Syndrome and key parts h_i are stored in BRAMs and processed as 151 32-bit words. The chance of a 32-bit key word to be all-zero 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.

Independent of the ciphertext input x, the stored key row $\langle h_0 \lll i || h_1 \lll i \rangle$ 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 [von Maurich and Güneysu, 2014]). The implementation features a single carry bit, stored in a separate register. In each of these clock cycles, one bit $h_{i,j}$ is written to a carry register, causing leakage $\lambda_{j,\text{carry}}$. In the following clock cycle, that bit is overwritten with $h_{i,j+32}$. Assuming a Hamming distance leakage function, this register leaks first

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

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

Besides the key rotation, the computation of the syndrome s contributes significantly to the leakage. The target implementation processes the ciphertext x in a bitwise fashion. If the i-th bit is set, i. e., $x_i = 1$, then the i-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 each clock cycle. This means that the addition of one row of H takes 151 clock cycles (plus two additional clock cycles for preprocessing and data read-write delay, resulting again in 153 clock cycles). The syndrome s is only updated if at least one of the currently processed ciphertext bits x_i is set.

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 and x, it is predictable which 32-bit word of h_i is added to the syndrome at which point in time. Similarly, it is predictable which key bit h_i enters the carry register in which clock cycle for the key rotation. We use this information to build the differential power analysis attacks.

As in the classical DPA by Kocher et al., we can now hypothesize each key bit $h_{i,j}$ separately to be one, knowing that this hypothesis will be wrong 99% of the time. We further know at which clock cycle the leakage of the carry register (for the key rotation) *or* the currently written syndrome word (for the computation of the syndrome) depends on $h_{i,j}$. Based on this knowledge, one can build two different attacks:

3.2 DPA of Syndrome Computation

The first analysis targets the leakage of the syndrome during its computation. This analysis assumes the adversary to send chosen ciphertexts of weight one, i. e., all possible x such that $\text{wt}(x) = 1$. Ciphertexts of weight 1 ensure that a rotated version of either h_0 or of h_1 is written into a zeroed syndrome s. To recover h_0, we chose only the first 4801 bits of x to be one, yielding a total of 4801 different ciphertexts for the analysis.[4]

For each x we know when a line of the key is added to the syndrome. We also know at which clock cycle during that addition the word containing $h_{i,j}$ is added. Our algorithm recovers the clock cycle where the $h_{i,j}$ is added to s for each x and the corresponding leakage in the leakage trace L. Next, we simply sum all the leakage instances of the target $h_{i,j}$ for the different x_i into a bin, as typically done by DPA. Unlike DPA, we have only one bin per key bit. However, assuming that each bit leaks similarly, we have 4756 bins that correspond to a $h_{i,j} = 0$, and only 45 bins corresponding to a bit $h_{i,j} = 1$.

The leakage model for the registers containing the syndrome is described next. Since the ciphertexts are of weight one, the syndrome is initially zero, and then overwritten with (a shifted version of) h_0 or h_1. The key bit $h_{i,j}$ is processed as part of one 32-bit word $\langle h_{i,j-l} \dots h_{i,j} \dots h_{i,j-l+31} \rangle$, where $l \in \{0, \dots, 31\}$ depends on j and the position of the set bit in x. As mentioned before, this leakage occurs during a specific cycle that we can reconstruct. Assuming a Hamming distance leakage, the Hamming weight of the word will leak, since it overwrites a zeroed register, i. e., the leakage of the corresponding syndrome word can be modeled as

$$\lambda_{j,\text{syn}} = w_0 \cdot \text{wt}\left(\langle h_{i,j-l} \dots h_{i,j} \dots h_{i,j-l+31} \rangle\right)$$

with an appropriate weight $w_0 \in \mathbb{R}$.

This approach has two disadvantages, one being the inaccuracy of the model; the other disadvantage is that bits of h_i located close to each other have highly correlated leakage functions. In fact, since 32-bit registers are leaking, all bits in the same register will enter the leakage function in the same way. However,

[4] As detailed in Section 5, once h_0 is known the remaining part of the secret key can be derived easily.

whether a given neighboring bit is in the same register depends on the row index that is currently processed, since the key bits are rotated by one bit for each row. This means that the neighboring bits will leak in a different clock cycle eventually, as the position of the set bit in x changes for different ciphertexts. The closer the bit is to the correct bit, the higher their correlation is (since they are more likely to be in the same register). We will later show that, while key bits equal to 1 can be detected, their exact position is harder to detect, since neighboring bits "look like" ones as well.

The other problem are correlated leakages from the key rotation. Both h_0 and h_1 are rotated during the above computation, with the same key words being processed in the studied clock cycle. We detail on this leakage in Section 3.3. Since those leakages are dependent on the predicted bit, they are not independent noise that decreases by averaging, as usually happening in DPA. However, they occur independently of whether the syndrome is updated or not. It is possible to remove these leakages, which we refer to as $\lambda_{j,\mathrm{const}}$ by subtracting the average leakage during the corresponding clock cycles, i. e., the leakage of the same clock cycles when the key word is not added to the syndrome word (and the set bit in x is zero), given as

$$\mathcal{L}_{j,\mathrm{syn}} = \lambda_{j,\mathrm{syn}} + \lambda_{j,\mathrm{const}} + \mathcal{N},$$

where $\mathcal{N}$ is the noise, which is assumed to be Gaussian and can be minimized by increasing the number of observations used for computing $\mathcal{L}_{j,\mathrm{syn}}$. We can approximate $\lambda_{j,\mathrm{const}}$ by simply averaging over all observed traces and compute it as $\mathcal{L}_{j,\mathrm{const}} = \mathrm{avg}(L_j)$. This average is then subtracted from the leakage trace for $\mathcal{L}_{j,\mathrm{syn}}$, which is computed as

$$\Delta_{\mathrm{syn}}(j) = \sum_{l=0}^{4800} \left(\mathcal{L}_{j,\mathrm{syn}}(l) - \mathcal{L}_{j,\mathrm{const}}(l)\right). \qquad (2)$$

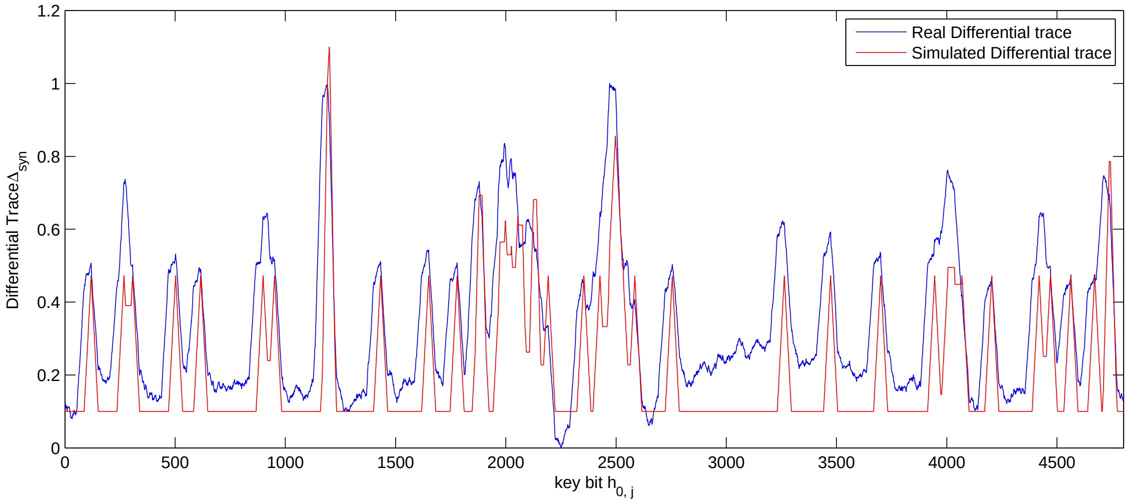


Fig. 2. Differential leakage for syndrome computation with key part h_0 only. The plot shows the normalized leakage (vertical axis) for each key bit of h_0 (horizontal axis) for simulated leakage according to $\lambda_{j,\mathrm{syn}}$ (blue/black line) and real measurement, i. e., empirical $\Delta_{\mathrm{syn}}(j)$ (red/gray line).

The resulting differential trace $\Delta_{\mathrm{syn}}(j)$ is depicted in Figure 2, where the red (gray) line depicts the observed leakage while the blue (black) line depicts the leakage derived from the model as described above. The plot of the differential trace shows the highest consumption for the correct key bits. The consumption decreases linearly as the distance to the bit increases, at least for key bits with a higher index. Bits at least 32 positions away from a set key bit show the lowest consumption, since they never share a leakage with a set bit. However, there is still a correlated leakage occurring that is not caught by our model. In fact, bits up to 64 bits lower than the predicted one still exhibit a correlation. We assume this to be due to the ReadFirst mode of the BRAM. In fact, when a specific syndrome word is written to BRAM, the next one is simultaneously read, as is the corresponding part of the key. Hence, the next clock cycle's word could already be computed. While we expect this leakage to be constant, i. e., to occur independently of whether the syndrome will be updated or not, the observed leakage suggests otherwise.

As a summary, the described method lets us detect leakages of h_0 and h_1 separately. It allows us to reliably distinguish set bits from zero bits. We get a single leakage observation per trace L for chosen ciphertexts of weight 1. However, closely co-located bits are highly correlated, making the exact position of a bit difficult to detect.

3.3 DPA of Key Rotation

The next analysis targets the key rotation itself. Our algorithm identifies all clock cycles where $h_{i,j}$ is written to or overwritten in the carry register in each trace 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. By identifying the corresponding clock cycles and adding them together, one can generate a differential power trace $\mathit{\Delta}_{\text{carry}}$, as typically done by DPA. As for the DPA of the syndrome computation, we have only one bin per key bit. Since the key is sparse, we have only very few bins that correspond to a $h_{i,j} = 1$, while most bins correspond to a bit $h_{i,j} = 0$. The target implementation processes h_0 and h_1 in parallel. This means that 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 do not attempt to distinguish them. Instead we recover the combined leakages. That is, we predict the combined leakage $h = h_0 + h_1$. 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.

The 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 addition to the leakage of the carry register $\lambda_{j,\text{carry}}$ 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 leaks as

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

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 by the following model:

$$\mathcal{L}_j = \lambda_{j,\text{carry}} + \lambda_{j,\text{read}} + \lambda_{j,\text{store}} + \mathcal{N}$$

where $\mathcal{L}_j$ is the overall leakage at the clock cycle where $h_{i,j}$ is written into the carry register and $\mathcal{N}$ is the noise, which is assumed to be Gaussian. While the model is not perfect, it describes the observed leakages well enough to base a decent key recovery on it.

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.

$$\mathit{\Delta}_{\text{carry}}(j) = \frac{1}{150} \sum_{l=1}^{150} \mathcal{L}_{j,l} = \text{avg}\left(w_1 \cdot \text{wt}(h_{i,j-32} \oplus h_{i,j}) + \lambda_{j,\text{read}} + \lambda_{j,\text{store}}\right)$$

Note that, since $h_{i,j-32} = 0$ with very high probability, $\Delta_{\text{carry}}(j)$ depends directly on the key bit. Further note that $h_{i,j} = 1$ has an even stronger influence on $\Delta_{\text{carry}}(j \pm 32)$, since it leaks through $\lambda_{j,\text{carry}}$ and either $\lambda_{j,\text{read}}$ or $\lambda_{j,\text{store}}$. The dependence of $\Delta_{\text{carry}}(j)$ on neighboring key bits $h_{i,j\pm l}$, with $l \leq 32$, implies that each set key bit not only results in an increased differential 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 differential trace. These shapes can (and actually will) overlap if several key bits in the same region are set.

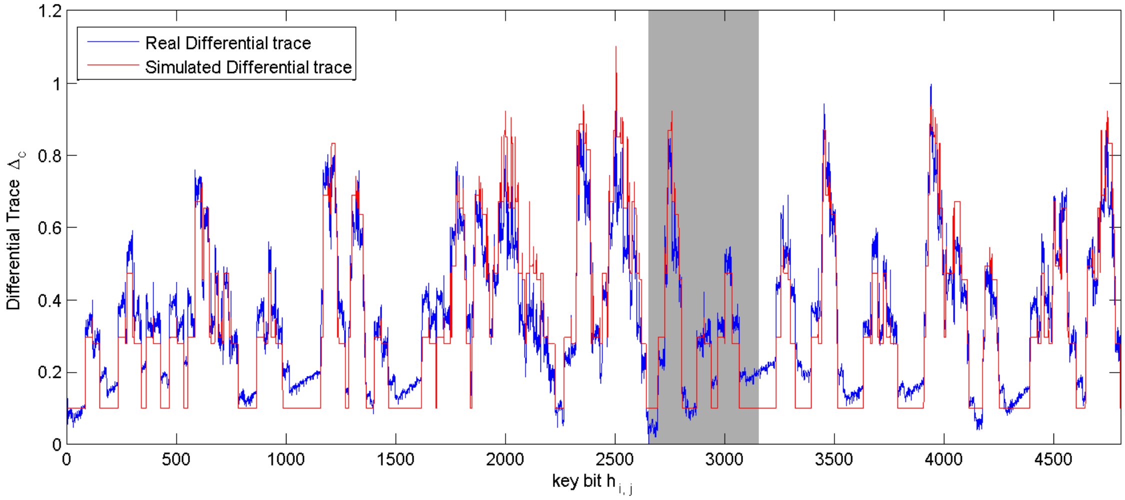


Fig. 3. Differential trace for key rotation. The plot shows the normalized leakage (vertical axis) of both key parts $h = h_0 + h_1$. The red/gray line is the simulated leakage while the blue/black line is the observed leakage from the target implementation.

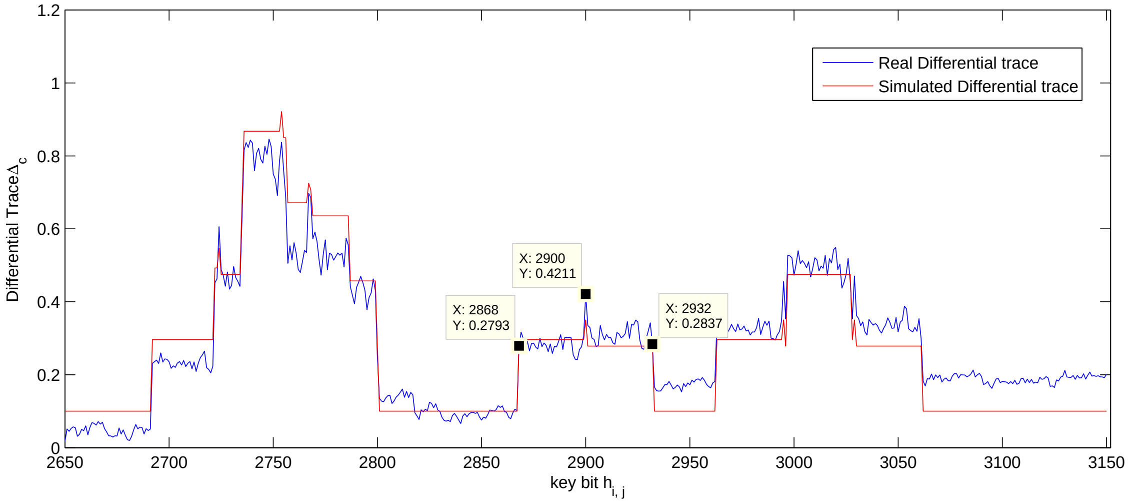


Fig. 4. A magnified version of Figure 3 that highlights the characteristic shape of a single set bit (center) as well as the overlap of two (right) and three (left) "adjacent" set bits.

Figure 3 shows the comparison of the simulated differential trace (red/gray line) using the power model and the real differential trace (blue/black line). The characteristic shape is highlighted in Figure 4, which is a magnification of a single set bit of the key, surrounded by zeroes.

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. Unlike the leakage of the syndrome computation, the key rotation leakage does not feature a slowly decreasing correlating leakage, but rather a characteristic shape with easily detectable bounds. This allows for a more 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.

3.4 Key Bit Recovery

The computation of syndrome and key rotation both cause leakages which can be analyzed in the presented differential traces. In both of the differential traces, 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.

For the computation of the syndrome, the differential trace can recover the key bits of h_0 or h_1 separately, depending on the ciphertext we use. For the key rotation, since the analyzed implementation processes h_0 and h_1 in parallel, resulting in an overlap of the leakages, the differential trace actually recovers the key bits of $h = h_0 + h_1$.

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 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 4. 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 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 differential 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 [von Maurich and Güneysu, 2014] to a Xilinx Virtex-5 LX50 FPGA which is mounted on a Sasebo-GII side-channel attack evaluation board[5]. The implementation is clocked at 3 MHz. Measurements were performed using a Tektronix DPO 5104 oscilloscope at a sampling rate of 100 MS/s. Since all of our attacks focus 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.

As mentioned in Section 3, key rotation and syndrome computation run in parallel which leads to a mixed leakage. To fully exploit the leakages, measurements were obtained in three different scenarios:

[5] The VHDL code of the QC-MDPC McEliece implementation of [von Maurich and Güneysu, 2014] is available at `http://www.sha.rub.de/research/projects/code/`.

Known Ciphertext In this scenario we assume the adversary to only observe ciphertext-leakage pairs. Hence, the ciphertexts x are chosen uniformly at random. While this can result in invalid ciphertexts, the attacker could also just generate valid ciphertexts by choosing plaintexts at will. In this scenario, a mixed leakage of key rotation and syndrome computation is obtained.

All-Zero Ciphertext In order to minimize the impact of the syndrome computation and storage on the leakage, we recorded the power consumption for an all-0 ciphertext. The syndrome is never updated when the ciphertext is 0, while key rotation is always executed. Note that the all-zero word is a valid codeword without any errors. This corresponds to a chosen ciphertext side-channel attack, without the need to observe the corresponding plaintext.

Single-One Ciphertext As mentioned in Section 3.2, the ciphertext weight is chosen to be one in this scenario, i. e., only a single bit of the ciphertext is set. This is done by adding a one bit error in each position of the all-0 ciphertext. There are 9602 such ciphertexts since both message and the redundant part have 4801 bit positions.

4.1 DPA Results of the Syndrome Computation

To extract key leakage from the syndrome computation, the single-1 ciphertexts give the main contribution. In fact, they provide the leakages of the $\mathcal{L}_{j,\mathrm{syn}}(l)$ term in Equation (2). The syndrome-storage independent leakage $\mathcal{L}_{j,\mathrm{const}}(l)$ can either be derived by an average of several all-0 leakage traces or the average of all used single-1 measurements. The latter approach has the advantage of not requiring additional measurements. We chose the former approach, as it is slightly less noisy. By subtraction of the two leakage terms, we derive the leakage of the syndrome computation only. Figure 5 shows the differential trace of the syndrome computation with respect to h_0.

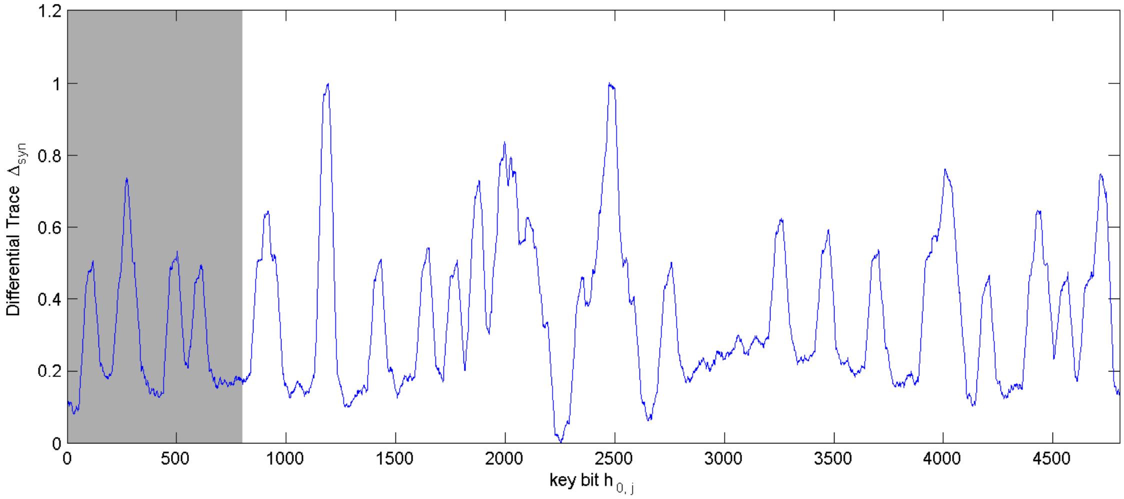


Fig. 5. Differential trace for syndrome computation $\Delta_{\mathrm{syn}}(j)$ for h_0 only. Peak shapes correspond to set bits in the key part h_0. Due to correlation in the leakage of closely located bits, the shapes overlap on several positions.

The magnification of the differential trace in Figure 6 highlights the observed characteristic shapes imprinted by set key bits $h_{0,j} = 1$. The shape on the left is caused by a single set key bit $h_{0,118}$ with neighboring key bits set as 0. The shape on the right is the result of two overlapping shapes of set bits in position 267 and 306, i. e., $h_{0,267} = h_{0,306} = 1$.

Key Extraction To actually recover the key bits from the differential trace $\Delta_{\mathrm{syn}}(j)$, the recovery algorithm described in Section 3.4 is applied. The first step is to build the threshold based on features in the shape.

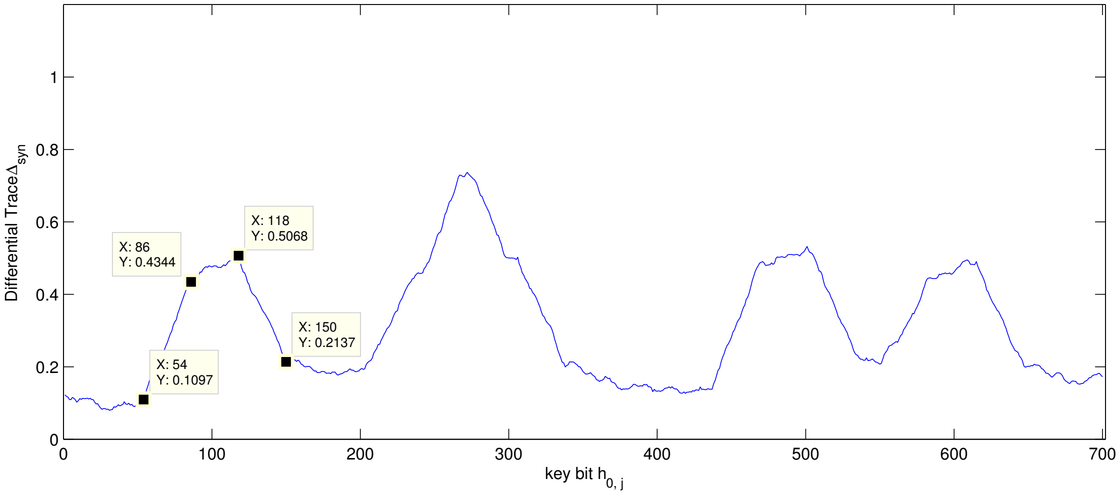


Fig. 6. Magnification of the characteristic shape of a single set key bit (left, $h_{0,118} = 1$) and two adjacent set key bits (center left, $h_{0,267} = h_{0,306} = 1$). The two shapes on the right are due to two other set key bits ($h_{0,501} = 1$ and $h_{0,616} = 1$).This plot is a magnification of the gray shaded area of Figure 5.

As shown in Figure 6, the set key bit $h_{0,j} = 1$ for $j = 118$ caused a characteristic shape where there are two strong features. One is a rising slope from $h_{0,j-64}$ to $h_{0,j-32}$ and the other one is a falling slope from $h_{0,j}$ to $h_{0,j+32}$.

An easy way to detect slopes is by computing the backward difference of $\Delta_{\text{syn}}(j)$ as $\Delta'_{\text{syn}}(j) = \Delta_{\text{syn}}(j) - \Delta_{\text{syn}}(j-1)$, which is strictly positive for rising slopes and strictly negative for falling slopes. The number of values for which $\Delta'_{\text{syn}}(j-64)$ to $\Delta'_{\text{syn}}(j-32)$ is positive and for which $\Delta'_{\text{syn}}(j)$ to $\Delta'_{\text{syn}}(j+32)$ is negative are counted separately. If both of the features exist, $h_{0,j}$ is taken as 1. If none of the features exist, $h_{0,j}$ is taken as 0. Otherwise, it is taken as undetermined. As discussed in Section 3.4, due to the overlapping and noise in the differential trace, there are false positive errors in the recovered key bits. The detection works very well for set key bits that are surrounded by zeros, and less well for set bits that are located close to each other. A partial improvement can be achieved by removing (subtracting) the leakage of detected bits from the leakage trace and thereby decomposing an area of overlapping shapes into its components. However, this process turned out to be quite error-prone in itself, so that we did not further explore that direction. As we show in Section 5, such improvements to the detection algorithms are not necessary, as the recovered information is already plenty to recover the correct key.

Table 1. Key bit recovery rates (#rec) and bit error rates (#error) for h_0 based on the leakage of the syndrome computation for various thresholds and number of traces.

Key bit	Total # of	Threshold: 16		Threshold: 20		Threshold: 24		Threshold: 28	
value	traces	#rec	#error	#rec	#error	#rec	#error	#rec	#error
0	$1 \cdot 4801$	2636	0	3281	4	4089	12	4702	34
	$2 \cdot 4801$	2672	0	3143	2	3749	6	4463	17
	$5 \cdot 4801$	2681	1	3063	3	3573	6	4133	10
	$10 \cdot 4801$	2703	0	3035	3	3439	6	3931	8
1	$1 \cdot 4801$	14	12 (0)	10	7 (0)	3	2 (0)	0	0(0)
	$2 \cdot 4801$	32	25 (1)	17	13 (0)	11	8 (0)	3	2(0)
	$5 \cdot 4801$	137	118 (13)	74	59 (2)	30	21 (1)	8	5(0)
	$10 \cdot 4801$	248	225 (1)	166	145 (0)	76	60 (2)	26	15(0)

Table 1 shows the results using this recovery algorithm. For each experiment, a multiple of 4801 single-1 ciphertexts are used for computing $\Delta_{\text{syn}}(j)$. As expected, a lower threshold reduces the number of detected zeros, while it increases the number of detected ones. However, with a higher number of detections, the number of false positives usually goes up as well. Finally, a higher number of observed traces reduces noise and helps a cleaner shape detection. This is directly obvious from the zero recovery results, where the number of errors goes down for an increased number of used measurements. For the 1 recovery, the obvious improvement for more observations is the higher number of recovered bits. However, the number of false positives also tends to go up quickly with more measurements. This is due to the correlation effect for closely located bits described in Section 3.2. The described detection based on thresholds favors the detection of correlated bits close to true one bits as well. This means that the detected errors are bits located close to a true 1. In fact, for lower thresholds, the method returns sequences of ones, of which only one (of the center ones) is a true poitive. This means that for each set key bit there will be a few false positives in the neighboring bits as well. One could say that the ones are correctly detected, but that there is remaining uncertainty of the exact location. The number in the parentheses shows the number of false positives that cannot be explained by this, i. e., false positives that are not due to the choice of the threshold. We will later see that the remaining errors in the leakage can be fixed in the final full key recovery phase in Section 5.

4.2 DPA Results of the Key Rotation

Next, the results for the DPA of the key rotation are discussed. 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-zero ciphertext of which all bits are set as 0 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, where each bit in x is set with a 50% probability. For each scenario we took 256 measurements for various different keys. Note that, in the second case, a new random ciphertext is generated for each measurement.

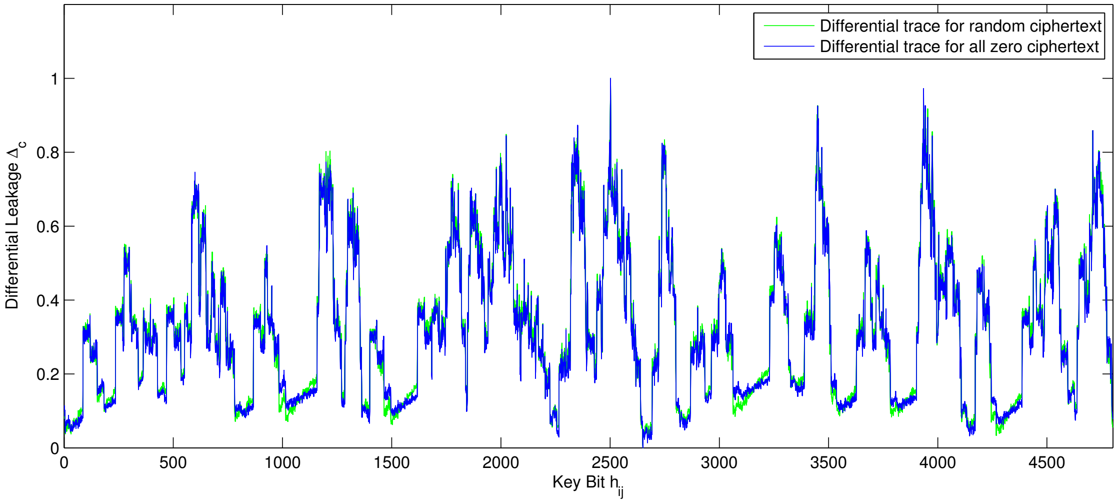


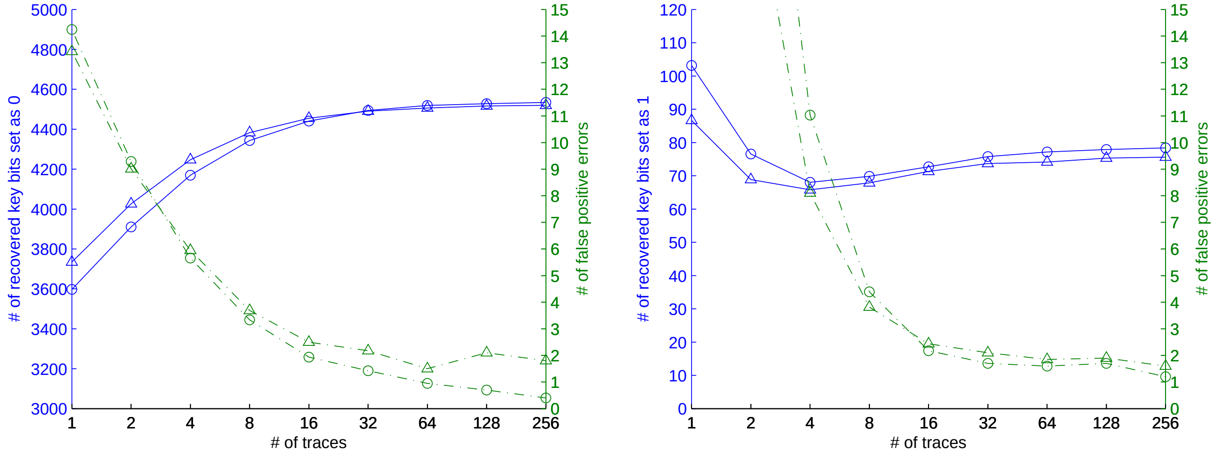
Fig. 7. Normalized differential leakage trace Δ_c for the key rotation for the bits of $h = h_0+h_1$. Whether the ciphertext is known (green/gray line) or all-0 (blue/black line) has only marginal influence on the observed leakage.

Next, we simply performed averaging over all considered traces in both scenarios. From the resulting average trace, $4801 \cdot 150$ peaks are extracted and used to construct the differential traces Δ_c as explained in

Section 3.3. Note that averaging explicitly before the computation of Δ_c or implicitly during the computation of Δ_c does not influence the result. Figure 7 shows the differential traces for the key rotation, showing the key bit position (horizontal axis) vs. the differential power (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 4 shows magnifications of the differential trace to highlight the characteristic shapes, particularly the one generated by setting the key bit $h_{i,2901}$ as 1 and the neighboring key bits as 0. The other shapes in Figure 4 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 as easily be distinguished, we did not further try to exploit this information.

Key Extraction To extract keys from Δ_c, we used the algorithm described in Sec. 3.4. 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 7 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_c, 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_c, 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 8 shows the comparison of number of recovered key bits and false positive errors between the all-zero ciphertext and random ciphertext. In Figure 8.1, as the number of traces used to generate the differential 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-zero ciphertext, there are less positive errors. Figure 8.2 shows the comparison of the number of recovered key bits of 1 and the false positive errors. It can be seen that using the all-zero ciphertext recovers more key bits of 1 with fewer false positive errors as the number of traces increases. In conclusion, the all-zero ciphertext is more advantageous to the DPA of key rotation. Hence, we use the traces with the all-zeros ciphertext in the following experiment. Note that we used five different keys to validate the conclusion and thus the figure actually shows the average result of five keys.

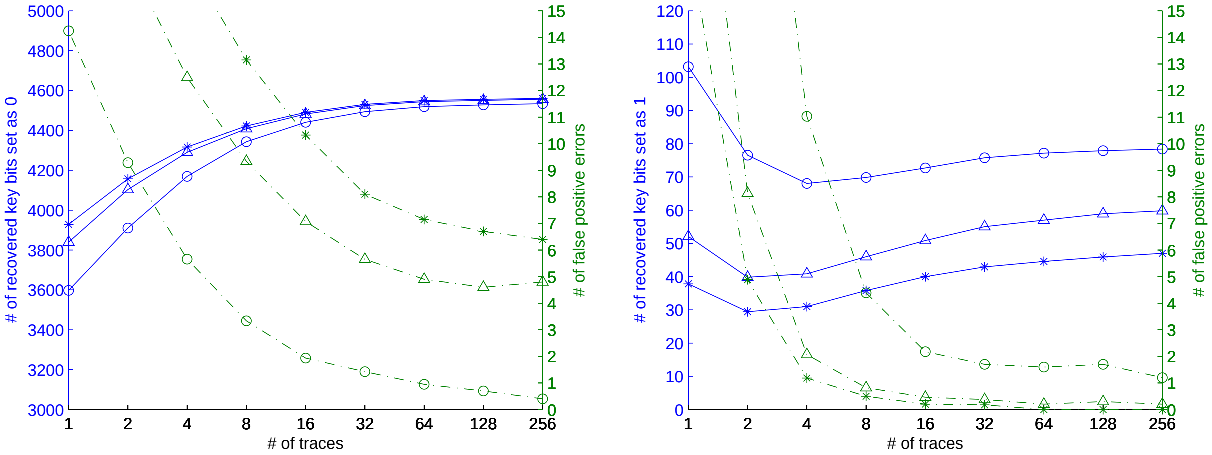


8.1: Recovered 0 key bits vs. false positives.

8.2: Recovered 1 key bits vs. false positives.

Fig. 8. Key bit recovery rates for *known* random (○) vs. *chosen* all-0 (△) ciphertext 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 right), the dashed line indicates the number of false positives (scale on left).

Figure 9 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_c. The other two ($\triangle$ and then $*$) are increased based on the first one. In Figure 9.1, as the number of traces used to generate the differential 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 9.2 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 least false positive errors, the more aggressive threshold should be used.



9.1: Recovered 0 key bits vs. false positives.

9.2: Recovered 1 key bits vs. false positives.

Fig. 9. 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 right), the dashed line indicates the number of false positives (scale on left). Markers $\circ$, then $\triangle$, and then $*$ indicate the increasing values for the threshold.

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 already error-free with high probability. 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. Furthermore, unlike the leakage of the syndrome computation, the characteristic shape of the key rotation favors precise localization and straightforward error-minimizing detection.

5 Full Key Recovery

Next we analyze how to recover the full key of QC-MDPC McEliece in a scenario where 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:

$$h_0 = h_1 \cdot Q^T \tag{3}$$

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 $\mathrm{wt}(h_{\bar{i}}) = 45$. Unfortunately, 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.

In one of the attacks described above the adversary observes a combined leakage of h_0 and h_1. 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 (3) we obtain

$$h_0 \oplus h_1 = h_1 \cdot (Q^T \oplus I_{4801}). \tag{4}$$

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 [Bosma et al., 1997]—and then derive h_0 from Equation (3). 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 (3) and (4) enable a practical key recovery.

The syndrome computation leakage provides information on the positions of ones in h_0 or h_1, but 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 as well as several 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

$$B_0 \,\dot\cup\, B_1 \,\dot\cup\, B_u = \{0, 1, \ldots, 4800\}\,. \tag{5}$$

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.[6] Hence, the uncertain positions for h_1 are $B_u^1 = B_1 \,\dot\cup\, B_u$, and with Iverson's convention [Knuth, 1992] we can summarize our knowledge of $h_0 \oplus h_1$ and h_1 as

$$h_0 \oplus h_1 = \langle 1 \cdot [i \in B_1] + u \cdot [i \in B_u] \rangle_{0 \le i \le 4800} \text{ and}$$
$$h_1 = \left\langle u \cdot [i \in B_u^1] \right\rangle_{0 \le i \le 4800},$$

where u indicates unknown bits ("erasures"). So Equation (4) yields

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

[6] 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.

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

$$\langle 1 \cdot [i \in B_1] + u \cdot [i \in B_u] \rangle_{0 \leq i \leq 4800} = \left\langle u \cdot [i \in B_u^1] \right\rangle_{i \notin B_0} \cdot Q'. \tag{6}$$

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 (6). 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

$$\langle 1 \cdot [i \in B_1] \rangle_{i \notin B_u} = \left\langle u \cdot [i \in B_u^1] \right\rangle_{i \notin B_0} \cdot Q'' \tag{7}$$

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

To check the practical applicability and feasibility of this approach, we ran several experiments in Magma [Bosma et al., 1997], solving the equation system given in (7) 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 (5) we see that the condition $|B_u| \leq |B_0|$ now implies that $|B_0| \geq \lceil 4801/2 \rceil$. 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 (7) 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 positives among the $|B_0|$ indices, the probability of guessing 2401 error-free positions is

$$\frac{\binom{|B_0|-f}{2401}}{\binom{|B_0|}{2401}}.$$

For instance, with $|B_0| = 3281$ and $f = 4$, this probability is still about $2^{-7.6}$.

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.

Remark 1. If the target platform allows the separation of leakages for h_0 and h_1, the above strategy naturally carries over when Equation (3) (instead of Equation (4)) is used as starting point.

6 Preventing the Attacks

The described attacks are 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 attacks further. However, a massively parallelized implementation such as the one described in [Heyse et al., 2013] could impede the described attacks, since all bits would always be leaking in parallel. One might however 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 these attacks are side-channel countermeasures. A good overview of standard DPA countermeasures is available in [Mangard et al., 2007]. Countermeasures are typically classified as *masking* or *hiding* countermeasures. Both classes can be applied to an implementation of MDPC McEliece and, if done correctly, should prevent the above-mentioned attacks. Note that masking would need to be applied to the syndrome *and* the key, since the presented attacks target both leakage sources separately. The disadvantage of SCA countermeasures is usually the large performance overhead they introduce. Masking, for example, typically doubles the amount of storage needed for internal states. So far, SCA countermeasures for (QC-)MDPC McEliece have not been studied in detail in the literature, making it an interesting subject for future work.

To spur the discussion, we point out a plausible solution strategy that should prevent the introduced attacks while maintaining a comparably low footprint. All of the described attacks take advantage of the knowledge of *when* a specific key bit is processed. This advantage only holds for deterministic execution orders. A countermeasure that randomizes the execution order is known as *shuffling* and has been discussed in detail, e. g., in [Tillich and Herbst, 2008]. Shuffling the syndrome computation is fairly simple. Ciphertext bits and key bits could be processed in a random order. This would require the implementation to be able to rotate the private key and possibly the syndrome by various offsets while ensuring that these offsets are not detectable by the adversary. Implementing shuffling in such a way that no additional leakages are introduced is not a trivial task, as discussed in [Veyrat-Charvillon et al., 2012], for instance. However, such an implementation can be realized with comparably low area overhead, since no new arithmetic units nor additional storage, e. g., for masks, would be required. Furthermore, common counterattacks such as *combing* (cf. again to [Tillich and Herbst, 2008]) would not be helpful in this scenario, since it would require a summation over all clock cycles, making all key bits leak in parallel and thereby making them indistinguishable.

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 that is commonly used to achieve CCA security. Two different leakages are exploited. Both exploited leakages occur during the syndrome computation step of the decryption. The leakage of the syndrome register gives information on the two secret key halves h_0 and h_1 separately, but is slightly vague in the exact position of set key bits. Thousands of chosen ciphertext traces are necessary for successful key recovery. The attack on the key rotation 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 the known public key, which can be exploited to ease the key recovery process. 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 German Federal Ministry of Economics and Technology (Grant 01ME12025 SecMobil). RS is supported by NATO's Public Diplomacy Division in the framework of "Science for Peace", Project MD.SFPP 984520.

Bibliography

Roberto Avanzi, Simon Hoerder, Dan Page, and Michael Tunstall. Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems. *Journal of Cryptographic Engineering*, 1(4):271–281, 2011.

Elwyn R. Berlekamp, Robert J. McEliece, and Henk C.A. van Tilborg. On the Inherent Intractability of Certain Coding Problems (Corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, May 1978.

Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24:235–265, 1997.

Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic Cryptanalysis of McEliece Variants with Compact Keys. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Leture Notes in Computer Science*, pages 279–298, Berlin Heidelberg, 2010. International Association for Cryptologic Research, Springer.

Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, Frédéric de Portzamparc, and Jean-Pierre Tillich. Structural Cryptanalysis of McEliece Schemes with Compact Keys. Cryptology ePrint Archive: Report 2014/210, March 2014a. Available at `http://eprint.iacr.org/2014/210`.

Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, Frédéric de Portzamparc, and Jean-Pierre Tillich. Folding Alternant and Goppa Codes with Non-Trivial Automorphism Groups. Cryptology ePrint Archive: Report 2014/353, May 2014b. Available at `http://eprint.iacr.org/2014/353`.

Robert Gallager. Low-density Parity-check Codes. *Information Theory, IRE Transactions on*, 8(1):21–28, 1962.

Stefan Heyse, Amir Moradi, and Christof Paar. Practical Power Analysis Attacks on Software Implementations of McEliece. In Nicolas Sendrier, editor, *Post-Quantum Cryptography – PQCrypto 2010*, volume 6061 of *Lecture Notes in Computer Science*, pages 108–125, Berlin Heidelberg, 2010. Springer.

Stefan Heyse, Ingo von Maurich, and Tim Güneysu. Smaller Keys for Code-Based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 273–292, Berlin Heidelberg, 2013. Springer.

W. Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes.* Cambridge University Press, United Kingdom, 2010.

Donald E. Knuth. Two Notes on Notation. *The American Mathematical Monthly*, 99(5):403–422, May 1992.

Kazukuni Kobara and Hideki Imai. Semantically Secure McEliece Public-Key Cryptosystems –Conversions for McEliece PKC–. In Kwangjo Kim, editor, *Practice and Theory in Public Key Cryptosystems – PKC '01*, volume 1992 of *Lecture Notes in Computer Science*, pages 19–35, Berlin Heidelberg, 2001. Springer.

Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Berlin Heidelberg, 1999. Springer.

Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smartcards.* Springer, US, 2007.

Robert J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes. Cryptology ePrint Archive, Report 2012/409, 2012. `http://eprint.iacr.org/2012/409`.

Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory (ISIT)*, pages 2069–2073. IEEE, 2013.

Ryo Nojima, Hideki Imai, Kazukuni Kobara, and Kirill Morozov. Semantic security for the McEliece cryptosystem without random oracles. *Designs, Codes and Cryptography*, 49(1–3):289–305, December 2008.

Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms On a Quantum Computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. ISSN 0097-5397.

Abdulhadi Shoufan, Falko Strenzke, H.Gregor Molter, and Marc Stöttinger. A Timing Attack against Patterson Algorithm in the McEliece PKC. In Donghoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology – ICISC 2009*, volume 5984 of *Lecture Notes in Computer Science*, pages 161–175. Springer, Berlin Heidelberg, 2010.

Falko Strenzke. A Timing Attack against the Secret Permutation in the McEliece PKC. In Nicolas Sendrier, editor, *Post-Quantum Cryptography – PQCrypto 2010*, volume 6061 of *Lecture Notes in Computer Science*, pages 95–107, Berlin Heidelberg, 2010. Springer.

Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side Channels in the McEliece PKC. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography – PQCrypto 2008*, volume 5299 of *Lecture Notes in Computer Science*, pages 216–229, Berlin Heidelberg, 2008. Springer.

Stefan Tillich and Christoph Herbst. Attacking State-of-the-Art Software Countermeasures – A Case Study for AES. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 228–243. Springer, Berlin Heidelberg, 2008.

Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, Berlin Heidelberg, 2012.

Ingo von Maurich and Tim Güneysu. Lightweight code-based cryptography: QC-MDPC McEliece encryption on reconfigurable devices. In *Design, Automation and Test in Europe – DATE 2014*, pages 1–6. IEEE, 2014.

Carolyn Whitnall, Elisabeth Oswald, and François-Xavier Standaert. The Myth of Generic DPA...and the Magic of Learning. In Josh Benaloh, editor, *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *Lecture Notes in Computer Science*, pages 183–205, International Publishing, 2014. Springer.