A Side-Channel Assisted Cryptanalytic Attack Against QcBits

Mélissa Rossi[2,3,4(✉)], Mike Hamburg[1], Michael Hutter[1], and Mark E. Marson[1]

[1] Rambus Cryptography Research,
425 Market Street, 11th Floor, San Francisco, CA 94105, USA
{mike.hamburg,michael.hutter,mark.marson}@cryptography.com
[2] Thales Communications & Security, Paris, France
[3] Département d'informatique de l'ENS,
École normale supérieure, CNRS, PSL Research University,
75005 Paris, France
melissa.rossi@ens.fr
[4] INRIA, Paris, France

Abstract. QcBits is a code-based public key algorithm based on a problem thought to be resistant to quantum computer attacks. It is a constant-time implementation for a quasi-cyclic moderate density parity check (QC-MDPC) Niederreiter encryption scheme, and has excellent performance and small key sizes. In this paper, we present a key recovery attack against QcBits. We first used differential power analysis (DPA) against the syndrome computation of the decoding algorithm to recover partial information about one half of the private key. We then used the recovered information to set up a system of noisy binary linear equations. Solving this system of equations gave us the entire key. Finally, we propose a simple but effective countermeasure against the power analysis used during the syndrome calculation.

Keywords: QcBits · Post-quantum cryptography · McEliece · Niederreiter · QC-MDPC codes · Side-channel analysis · Differential power analysis · Noisy binary linear equations · Learning parity with noise

1 Introduction

1.1 Quantum Computers and Post-Quantum Cryptography

The security of the most commonly-used public key cryptosystems is based on the difficulty of either the integer factorization problem or the discrete logarithm problem. Unfortunately, both of these problems can be efficiently solved using quantum computers [36]. Progress in quantum computing has been steady, and many believe that practical quantum computers will become a reality

M. Rossi—This work was done while the author was at Rambus Cryptography Research.

W. Fischer and N. Homma (Eds.): CHES 2017, LNCS 10529, pp. 3–23, 2017.
DOI: 10.1007/978-3-319-66787-4_1

within the next 20 years [12,28]. In fact, the National Security Agency (NSA) and the National Institute of Standards and Technology (NIST) have both issued announcements calling for the standardization and transition to post-quantum public key algorithms in the near future [12,33]. A European initiative, PQCRYPTO, sponsored by the European Commission under its Horizon 2020 Program, published a report entitled "Initial Recommendation of long-term secure post-quantum systems" [1]. This report recommends the development of cryptography which is resistant to quantum computers. These concerns about quantum computers have given research in post-quantum cryptography a great deal of momentum in the past few years. Some of the most promising directions include cryptosystems based on lattices, error correcting codes, hash functions, and multivariate quadratic equations. The mathematical problems upon which these cryptosystems are based are expected to remain intractable even in the presence of quantum computers [7].

In this paper, we analyze and successfully attack a code-based post-quantum public key cryptosystem called QcBits [13]. QcBits (pronounced "quick-bits") is a variant of the McEliece public-key cryptosystem [24] based on quasi-cyclic (QC) moderate density parity check (MDPC) codes [26]. Although the McEliece cryptosystem in its original form is still regarded as secure, the public keys for the originally proposed parameters are very large. On the other hand, cryptosystems based on QC-MDPC codes have much smaller and simpler public and private keys. The quasi-cyclic form allows the public and private keys to be completely defined by the first rows of their matrices.

However, it is precisely the quasi-cyclic structure and moderate density of the private key which allows our attack to succeed. The QcBits secret parity check matrix is the concatenation of two sparse circulant matrices, denoted $\boldsymbol{H}_0$ and $\boldsymbol{H}_1$. We first used differential power analysis (DPA) against $\boldsymbol{H}_0$ to narrow down the locations of its nonzero elements. This gave us enough information to set up a system of noisy binary linear equations, which we could solve with high probability. Solving these equations gave us both the exact matrix $\boldsymbol{H}_0$, as well as the other matrix $\boldsymbol{H}_1$.

1.2 Previous Related Work

The first code-based public key cryptosystem is due to McEliece [24]. Its security is based on the difficulty of decoding a random linear code. It has been extensively analyzed since being proposed, and is still regarded as secure in its original form using Goppa codes. The main drawback of this construction is the size of the public keys. For the originally proposed parameters these keys contain about 500 Kbits. This drawback motivated the search for secure code-based cryptosystems with more manageable key sizes [19,23,29,38]. Unfortunately, most of the proposed McEliece variants using codes other than Goppa codes have turned out to be insecure [14,22,25,27,34,39]. Using QC-MDPC codes to replace Goppa codes in the McEliece cryptosystem was first suggested by Misoczki *et al.* in 2013 [26], and appears to be a promising choice. Some hardware implementations of this scheme followed in 2013 [18] and 2014 [42].

QC-MDPC codes are characterized by moderate density parity check matrices in quasi-cyclic form. The quasi-cyclic form allows both the public key and private key matrices to be completely defined by their first rows, leading to much smaller key sizes. Also, because of the way the public generator matrix is constructed, there is no need for scrambling and permutation matrices. Instead, the generator matrix is directly presented as a public key in its systematic form. In [1], the PQCRYPTO group recommends the QC-MDPC scheme for further study.

QC-MDPC McEliece was originally designed to be secure against chosen plaintext attacks (CPA) but not against chosen ciphertext attacks (CCA). To achieve security against adaptive chosen ciphertext attacks, some transformations were proposed in [4,20]. A hybrid CCA-secure encryption protocol using QC-MDPC Niederreiter was proposed by Persichetti [32] and implemented by Von Maurich *et al.* [43]. QcBits is an implementation of a variant of this protocol due to Chou in [13]. It operates in a constant time and has very good speed results and small keys sizes.

Another issue with the QC-MDPC cryptosystems is that they have a non-negligible probability of decryption failure, with the failure rate depending on the security parameters. The failure rate was around 10^{-7} in Misoczki *et al.* original proposal [26], and is even worse for constant-time decoders. In [16], Guo *et al.* take advantage of the decryption failures to recover the secret key of Misoczki's original version in minutes. Preliminary work was done to improve constant-time decoding algorithms in [10], but they did not improve the failure rate below 10^{-7}. For CCA-secure versions of QC-MDPC cryptosystems, Guo *et al.* proposed a more complex version of their attack that requires at most 350 million decryptions and has a time complexity of $2^{39.7}$. QcBits is CCA-secure but it has a more advanced constant-time decoder [13]. Chou claims a failure rate of 10^{-8} for the 80-bit secure version. Guo *et al.* still estimate the time complexity for attacking QcBits to be $2^{55.3}$, but to our knowledge have not run the attack. They have not provided estimates against the 128-bit secure version. They proposed drastically reducing the decoding failure probability as countermeasure against this attack, but no details about how to do so have been published.

Side-channel attacks against code-based schemes have focused more on the original version of the McEliece cryptosystem based on Goppa Codes. Timing leakages were first studied in [41]. This was followed by Strenzke and Shoufan *et al.*, who performed a key recovery attack using timing analysis [37,40]. Heyse *et al.* performed a simple power analysis (SPA) attack against software implementations of classic McEliece algorithm [17]. In [11], Chen *et al.* describe a differential power analysis (DPA) [21] key recovery attack against a QC-MDPC FPGA McEliece implementation. To our knowledge, no DPA attacks have been performed on CCA-secure constant-time versions of QC-MDPC McEliece.

Our attack also includes solving a learning parity with noise (LPN) problem. We set up and solve a system of noisy binary linear equations to complete the key recovery. Solving such systems has a long history in cryptanalysis, with many different methods used depending upon the specifics of the problem.

See Belaid *et al.* in [2,3] for recent examples of such attacks. Our system of equations has very low noise. We therefore used an elementary method which, for very low noise systems (1%), was shown in [35] to be more efficient than the Blum-Kalai-Wasserman (BKW) algorithm [9].

1.3 Our Contribution

In this paper we present a side-channel assisted cryptanalytic attack against QcBits. In contrast to Guo *et al.*'s attack in [16], our attack focuses on the first step of the decoding process, and is independent of its failure probability. Our attack only requires us to observe a small number of decryptions (about 200 power traces for the implementation we analyzed), and we need to analyze less than 1% of each trace. Our attack also works for both the 80-bit and 128-bit security versions.

Our attack consists of two steps:

1. A DPA attack targeting the syndrome computation of the decryption operation. The operation uses half of the private key, and during this step we recover some information about that half of the key. Because of the way in which the implementation leaks, there is some ambiguity as to the exact location of the nonzero elements of the key.
2. A linear algebra computation which takes advantage of the sparseness of the private key and succeeds with high probability. We repeat this operation (varying the equations slightly each time) until the computation succeeds. This allows us to recover the entire secret key.

The number of traces required in the first step will of course depend upon the implementation and hardware on which it is run. The amount of work required for the second step will depend on how much information is recovered in the first step. For the implementation and hardware we used for our analysis, the DPA attack required about 200 power traces in Step 1. The work factors in Step 2 were 2^{24} for the 80-bit security version, and 2^{27} for the 128-bit security version. See Sect. 4 for details.

1.4 Paper Roadmap

In Sect. 2, we describe the QcBits cryptosystem introduced by Chou in [13]. In Sect. 3, we describe the DPA attack we used to recover information about the private key. In Sect. 4, we present the algebraic attack we implemented recovering the entire private key. In Sect. 5, we describe a simple countermeasure to help protect against our attack. Finally, in Sect. 6, we summarize our results and discuss future research.

2 Description of the QcBits Cryptosystem

2.1 Definitions

Definition 1 (Circulant matrix). *A $r \times r$ matrix is a* ***circulant matrix*** *if its rows are successive cyclic shifts of its first one.*

Definition 2 (Quasi-cyclic matrix). *A matrix* $\boldsymbol{H} = (\boldsymbol{H}_0, ..., \boldsymbol{H}_m)$ *is a* ***quasi-cyclic (QC) matrix*** *if the submatrices* $\boldsymbol{H}_0, ..., \boldsymbol{H}_m$ *are circulant matrices.*

Definition 3 (QC-MDPC code). *An* (n, r, w)-***QC-MDPC code*** *is a binary linear code with* n*-bit codewords and dimension* r *which is defined by a QC Moderate Density Parity Check (MDPC) matrix* $\boldsymbol{H}$*.*

$$\mathcal{C} = \{x \in \mathbb{F}_2^n | \boldsymbol{H} \cdot \boldsymbol{x}^T = 0\}. \tag{1}$$

In other words, the codewords are all the vectors in the right nullspace of $\boldsymbol{H}$ *which is QC and has a "moderate density". "Moderate" here means that* $\boldsymbol{H}$ *has a constant row weight* $w = O(\sqrt{n.\log(n)})$*.*

2.2 QC-MDPC Codes Used for QcBits

QcBits uses (n, r, w)-QC-MDPC binary codes with $n = 2r$. The parity check matrix in its QC-MDPC form is then composed of 2 square sparse circulant matrices

$$\boldsymbol{H} = (\boldsymbol{H}_0, \boldsymbol{H}_1) \in \mathbb{F}_2^{r \times n} \tag{2}$$

The generator matrix in its systematic form is the $r \times n$ binary matrix

$$\boldsymbol{G} = (\boldsymbol{I}, \boldsymbol{P}) \tag{3}$$

where $\boldsymbol{I}$ is the $r \times r$ identity matrix and $\boldsymbol{P}$ is an $r \times r$ dense binary circulant matrix

$$\boldsymbol{P} = (\boldsymbol{H}_1^{-1} \cdot \boldsymbol{H}_0)^T \tag{4}$$

The reader can easily verify that $\boldsymbol{H} \cdot \boldsymbol{G}^T = \boldsymbol{0}$, so the rows of $\boldsymbol{G}$ form a basis for the codewords. An r-bit data vector $\boldsymbol{x}$ is encoded by multiplying it by $\boldsymbol{G}$:

$$\boldsymbol{c} = \boldsymbol{x} \cdot \boldsymbol{G}. \tag{5}$$

Let $\boldsymbol{e}$ be a n-bit error vector, and $\hat{\boldsymbol{c}}$ the corrupted codeword

$$\hat{\boldsymbol{c}} = \boldsymbol{c} \oplus \boldsymbol{e} = \boldsymbol{x} \cdot \boldsymbol{G} \oplus \boldsymbol{e}. \tag{6}$$

In the general case, decoding a corrupted codeword (i.e., removing its errors) from a random binary linear code is an NP-hard problem [5]. However, if the QC-MDPC parity check matrix $\boldsymbol{H} = (\boldsymbol{H}_0, \boldsymbol{H}_1)$ is known and the Hamming weight of $\boldsymbol{e}$ is not too large, there are efficient algorithms for decoding corrupted QC-MDPC codewords. There is no known efficient algorithm if the two sparse circulant matrices $\boldsymbol{H}_0$ and $\boldsymbol{H}_1$ are not known. The most commonly-used decoding algorithm is the probabilistic bit-flipping algorithm introduced by Gallager in [15]. See Sect. 2.3 for details.

For the bit-flipping decoding algorithm on QC-MDPC codes, the maximum allowed number of bit errors, denoted t, is an estimated value. In [26] the authors determined values for QC-MDPC code parameters (n, r, w, t) which would provide

the desired security levels, while keeping the probability of a decoding failure as low as possible ($<10^{-7}$). The parameters they selected are shown in Table 1.

Table 1. Proposed QC-MDPC instances with security level

n	r	w	t	Bits of security
9602	4801	90	84	80
19714	9857	142	134	128

For the remainder of this paper, we focus on QC-MDPC codes with the two parameter sets (n, r, w, t) from Table 1. The private key of QcBits is the QC-MDPC parity check matrix $\boldsymbol{H}_{priv}$:

$$\boldsymbol{H}_{priv} = (\boldsymbol{H}_0, \boldsymbol{H}_1) \tag{7}$$

where $\boldsymbol{H}_0, \boldsymbol{H}_1 \in \mathbb{F}_2^{r \times r}$ are randomly generated circulant matrices with weight $\frac{w}{2}$ in each row. The private key is sparse, so only the indices of the nonzero values of the first row are stored. Knowing the private key, one can use the bit-flipping decoding algorithm to recover a codeword which has been corrupted by up to t errors.

The public key is computed directly from the private key $\boldsymbol{H}_{priv}$ as the dense circulant $r \times r$ matrix $\boldsymbol{P}$:

$$\boldsymbol{P} = (\boldsymbol{H}_1^{-1} \cdot \boldsymbol{H}_0)^T. \tag{8}$$

Knowing $\boldsymbol{P}$ allows anyone to build the generator matrix in its systematic form $\boldsymbol{G}_{pub}$ and a parity check matrix $\boldsymbol{H}_{pub}$:

$$\boldsymbol{G}_{pub} = (\boldsymbol{I}, \boldsymbol{P}), \tag{9}$$

$$\boldsymbol{H}_{pub} = (\boldsymbol{P}^T, \boldsymbol{I}). \tag{10}$$

2.3 QcBits Encryption and Decryption Algorithms

QcBits is an hybrid CCA-secure encryption protocol based on Niederreiter [29]. Unlike McEliece cryptosystem, Niederreiter uses the parity-check matrix rather than the generator matrix for the encryption. QcBits uses the following cryptographic primitives. See [13] for more details.

1. A hash function denoted $Hash$. QcBits uses Keccak [31];
2. A symmetric stream cipher denoted ($Senc$,$Sdec$). QcBits uses Salsa20 [8];
3. An authentication function denoted (Tag,$Check$). QcBits uses Poly1305 [6].

The encryption of a message $\boldsymbol{m}$ using QcBits is shown in Algorithm 1.

Algorithm 1. QcBits encryption

Data: Plaintext $\boldsymbol{m}$, Public matrix $\boldsymbol{P}$
Result: Ciphertext $(\boldsymbol{c}|\boldsymbol{d}|\boldsymbol{g})$
$\boldsymbol{e} \leftarrow \$$ // Drawing a random n-bit error vector with Hamming weight t
$\boldsymbol{key} \leftarrow Hash(\boldsymbol{e})$;
$\boldsymbol{c}^T \leftarrow (\boldsymbol{I}, \boldsymbol{P}^{-T}) \cdot \boldsymbol{e}^T \in \mathbb{F}_2^r$;
$\boldsymbol{d} \leftarrow Senc(\boldsymbol{key}, \boldsymbol{m})$;
$\boldsymbol{g} \leftarrow Tag(\boldsymbol{key})$;
Return $(\boldsymbol{c}|\boldsymbol{d}|\boldsymbol{g})$;

The reader can verify that $(\boldsymbol{c}|\boldsymbol{0}) \in \mathbb{F}_2^n$ is a codeword corrupted with the error $\boldsymbol{e}$. The encrypted message $\boldsymbol{d}$ has the size of the plaintext $\boldsymbol{m}$, as it is encrypted with a stream cipher. The message authenticator $\boldsymbol{g}$ is 16 bytes in length.

We next describe the bit-flipping algorithm, which is used by the decryption algorithm. Given a vector that is at most t errors away from a codeword, the bit flipping algorithm attempts to recover the codeword (or equivalently the error) using a sequence of iterations. During each iteration the algorithm decides which of the n positions of the input vector are most likely to be wrong, and inverts those bits. The resulting vector then becomes the input to the next iteration. In QcBits, the bit-flipping algorithm performs a total of $j_{max} = 6$ iterations. It uses the precomputed thresholds $Thresh[0, \ldots, 5] = [29, 27, 25, 24, 23, 23]$ in each iteration to determine which bits should be flipped. The bit-flipping process is shown in Algorithm 2.

Algorithm 2. Bit Flipping

Data: $\boldsymbol{H}_{priv} \in \mathbb{F}_2^{r \cdot n}, \boldsymbol{x} \in \mathbb{F}_2^n$
Result: Corrected codeword $\boldsymbol{v}$
$\boldsymbol{v} \leftarrow \boldsymbol{x}$;
$\boldsymbol{S} \leftarrow \boldsymbol{H}_{priv} \cdot \boldsymbol{v}^T$ // Syndrome computation;
for $j \in \{0, j_{max}\}$ **do**
  for $i \in \{0, ..., n-1\}$ **do**
    $\sigma_i \leftarrow \langle \boldsymbol{S}, \boldsymbol{h}_i \rangle \in \mathbb{Z}$ // $\boldsymbol{h}_i$ denotes the i-th column of $\boldsymbol{H}$;
    if $\sigma_i \geq Thresh[j]$ **then**
      $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i \oplus 1$
    end
  end
  $\boldsymbol{S} \leftarrow \boldsymbol{H}_{priv} \cdot \boldsymbol{v}^T$
end
Return the codeword $\boldsymbol{v}$

Algorithm 3 shows the decryption process. First, $(\boldsymbol{c}|\boldsymbol{0}) \in \mathbb{F}_2^n$ gets decoded. The bit-flipping returns the error $\boldsymbol{e}$. Then, the decryption hashes $\boldsymbol{e}$ to compute the symmetric key, verifies the tag $\boldsymbol{g}$, and decrypts the second part of the ciphertext, $\boldsymbol{d}$.

Algorithm 3. QcBits decryption

Data: Ciphertext $(\boldsymbol{c}|\boldsymbol{d}|\boldsymbol{g})$, Private key $\boldsymbol{H}_{priv} = (\boldsymbol{H}_0, \boldsymbol{H}_1)$
Result: Plaintext $\boldsymbol{m}$ or $\perp$
$\boldsymbol{s} \leftarrow (\boldsymbol{c} \mid \boldsymbol{0}) \in \mathbb{F}_2^n$;
$\boldsymbol{e} \leftarrow$ Bit-Flipping$(\boldsymbol{H}_{priv}, \boldsymbol{s}) \oplus \boldsymbol{s}$;
$\boldsymbol{key} \leftarrow Hash(\boldsymbol{e})$;
if $Check(\boldsymbol{key}, \boldsymbol{g})$ **then**
  Return $\boldsymbol{m} \leftarrow Sdec(\boldsymbol{key}, \boldsymbol{d})$
else
  Return $\perp$
end

We performed our side-channel attack against the use of the secret parity check matrix $\boldsymbol{H}_{priv}$ during Step 2 in Algorithm 2. This gave us enough information after just a few decryptions to complete the cryptanalytic attack. This is in contrast to the attack of Guo *et al.*, who obtained information about the key during the low-probability failures of Algorithm 3. We describe our attack in the next two sections.

3 Differential Power Analysis Attack Against QcBits

In this section, we describe how we used DPA to recover some partial information about the secret matrix $\boldsymbol{H}_0$. Our attack targets the syndrome calculation at the start of the bit-flipping algorithm, and recovers partial information about $\boldsymbol{H}_0$.

3.1 General Leakage Model

We analyzed the C code of QcBits and identified the syndrome computation of the bit-flipping decoding (Step 2 in Algorithm 2) as a candidate for a DPA attack:

$$\boldsymbol{H}_{priv} \cdot \begin{pmatrix} \boldsymbol{c}^T \\ \boldsymbol{0} \end{pmatrix} = (\boldsymbol{H}_0, \boldsymbol{H}_1) \cdot \begin{pmatrix} \boldsymbol{c}^T \\ \boldsymbol{0} \end{pmatrix} = \boldsymbol{H}_0 \cdot \boldsymbol{c}^T \tag{11}$$

where $\boldsymbol{c} \in \mathbb{F}_2^r$ is the first part of the ciphertext. We will focus our attention on this computation.

Let $\{x_0, ..., x_{(\frac{w}{2}-1)}\}$ denote the unknown indices of the nonzero elements of $\boldsymbol{h}_0$, the first row of $\boldsymbol{H}_0$. Because $\boldsymbol{H}_0$ is a circulant, it is uniquely defined by the x_i, and is represented in QcBits as a list of these indices. Due to its structure, the matrix $\boldsymbol{H}_0$ can be decomposed as a sum of $\frac{w}{2}$ rotation matrices

$$\boldsymbol{H}_0 = \boldsymbol{R}_{x_0} + ... + \boldsymbol{R}_{x_{(\frac{w}{2}-1)}}. \tag{12}$$

Multiplying $\boldsymbol{c}^T$ by $\boldsymbol{R}_{x_i}$, $0 \leq i \leq \frac{w}{2} - 1$, results in a left circular shift of $\boldsymbol{c}$ by x_i positions:

$$\boldsymbol{R}_{x_i} \cdot \boldsymbol{c}^T = \boldsymbol{r}_{x_i}(\boldsymbol{c})^T. \tag{13}$$

Hence the multiplication in Eq. 11 can be accomplished by computing the rotated ciphertexts $\boldsymbol{r}_{x_i}(\boldsymbol{c})$, $0 \leq i \leq \frac{w}{2} - 1$, and XORing them all together:

$$\boldsymbol{H}_0 \cdot \boldsymbol{c}^T = \bigoplus_{i=0}^{\frac{w}{2}-1} \boldsymbol{r}_{x_i}(\boldsymbol{c})^T. \tag{14}$$

In fact, this is how the multiplication is performed in the QcBits implementation. In a loop, each rotated vector $\boldsymbol{r}_{x_i}(\boldsymbol{c})$ is stored into a temporary memory location as it is calculated, and then XORed with the partial XOR sum from the previous loop iteration:

$$S_i = S_{i-1} \oplus \boldsymbol{r}_{x_i}(\boldsymbol{c}) = \bigoplus_{j=0}^{i-1} \boldsymbol{r}_{x_j}(\boldsymbol{c}) \oplus \boldsymbol{r}_{x_i}(\boldsymbol{c}). \tag{15}$$

Our side-channel analysis model assumes that the power consumption of the device depends on whether the leftmost bit (bit position 0) of each rotated vector $\boldsymbol{r}_{x_i}(\boldsymbol{c})$ is either 0 or 1 when it is stored to memory. Note that bit x_i of $\boldsymbol{c}$ is rotated into bit position 0 by $\boldsymbol{r}_{x_i}$ and into bit position 1 by $\boldsymbol{r}_{x_{i-1}}$. We therefore expect the device to leak for multiple guesses near the correct value, with the number of guesses exhibiting leaks related to the native word size of the device.

3.2 The Experiment Setup

We used the reference C version of QcBits[1] with 80 and 128 bits of security. We ported the code to run on ChipWhisperer evaluation platform designed by Colin O'Flynn [30]. The ChipWhisperer is a board composed of a programmable chip (Atmel AVR XMEGA128) and an on-board power-measurement circuit that can be connected to a PC via USB interface. An open-source python software is available that can be used to communicate with the chip, for example, to send encryption or decryption commands to the AVR. In order to measure the power consumption, the board features an analog to digital converter (OpenADC) that allows synchronous clocking to the AVR's clock. The clock frequency is fixed at 7.37 MHz. The signal is amplified with up to 55 dB gain and the power traces were sampled at a 96 MS/s rate.

We then generated a set of N known, random values $\{\boldsymbol{c}_0, ..., \boldsymbol{c}_{N-1}\} \in \mathbb{F}_2^r$. These were padded with zeros and passed to the bit-flipping Algorithm 2. Since they were randomly generated, the zero-padded values were almost certainly not codewords corrupted by at most t errors. As we were attacking the syndrome calculation at the beginning of the bit-flipping algorithm, however, we were not concerned with whether these values could be decoded properly. If properly formed ciphertext was required by the implementation, it could have been computed using the public-key information.

[1] Available at http://www.win.tue.nl/~tchou/qcbits/.

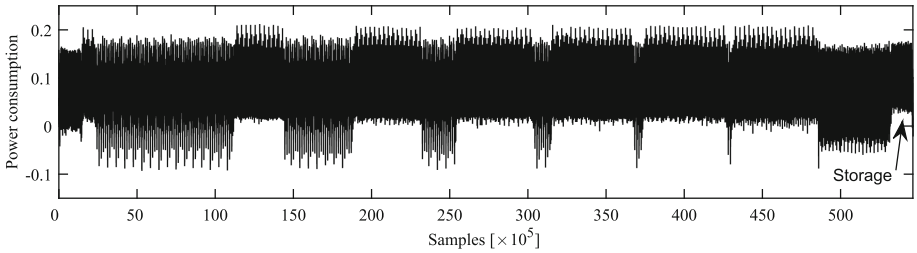

Fig. 1. Power trace of the first rotated ciphertext computation.

Figure 1 shows a typical power trace during the computation of one ciphertext rotation $\boldsymbol{r}_{x_i}(\boldsymbol{c})$ in QcBits. After the computation, the result is stored into memory, which can be seen in the power trace at the very end of the rotation operation. Figure 2 zooms into the store operation where the first 64-bits of the rotated value are written to memory. Because the XMEGA is an 8-bit architecture, we can observe eight different power patterns which are related to the storing of each 8-bit value from internal registers into internal RAM. We collected 13,000 traces of that operation for each key index, which was sufficient for our analyses. To characterize the leakage behavior of the device, we analyzed 25 different key indices, varying both the secret value and the loop iteration in which it gets XORed into the partial sum in Eq. 15.

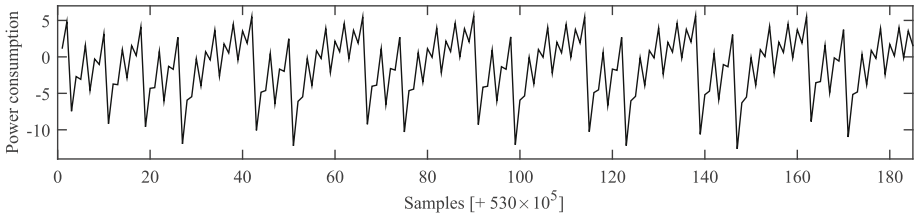

Fig. 2. Storing of the first 64 bits of the result of the rotation.

We attacked the unknown values $\{x_0, \ldots, x_{(\frac{w}{2}-1)}\}$ sequentially using standard DPA. We first made guesses for all possible values for the unknown x_0. Given the size of the secret matrix $\boldsymbol{H}_0$ this is clearly an exhaustible parameter. For each of those guesses, we sorted the traces T_j into two partitions based on whether the leftmost bit of the each rotated vector $\{\boldsymbol{r}_{x_0}(\boldsymbol{c}_0), ..., \boldsymbol{r}_{x_0}(\boldsymbol{c}_{N-1})\}$ was a zero or a one. We averaged the traces in the two partitions separately and computed the difference of the averages. Large spikes in the difference trace indicated a leak of information. As will be discussed in the next section, multiple guesses for each x_i exhibited significant leaks. This is due to how the algorithm was implemented, and how the hardware on the evaluation board leaked. We discuss how we resolved this ambiguity in Sect. 4. The DPA process is then repeated for each of the unknowns x_i.

3.3 DPA Results

Figure 3 shows the result of the DPA targeting for all possible values x_i using 500 power traces on the 80-bit version. The device clearly shows a significant leakage around the correct index (value 2,000 in this experiment). However, it also shows that there are other indices leaking, for example, the indices 1,985 up to 2,000 show similar Difference of Mean (DoM) values. We performed DPA attacks targeting other unknown indices of $\boldsymbol{h}_0$ and identified a particular leakage model. For a given secret index x_i, the device always leaks for 16 consecutive guesses starting at index

$$y_i = \lfloor \frac{(x_i - 1) \ mod \ r}{64} \rfloor \cdot 64 + 1, \tag{16}$$

which is $\lfloor \frac{2000-1}{64} \rfloor \cdot 64 + 1 = 1985$ in our example.

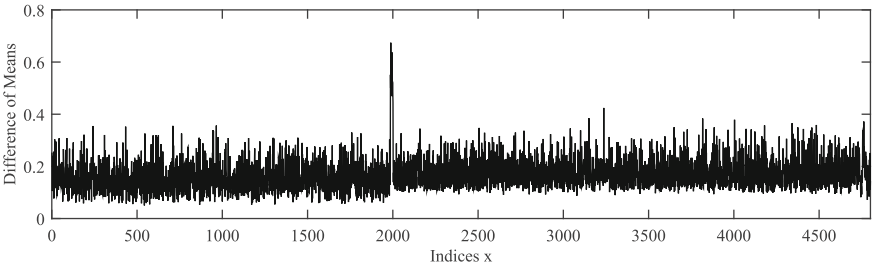

Fig. 3. Maximum Difference of Means (DoM) using 500 traces over all possible values x_i. Significant difference is observed for around the correct index 2000.

This gives us 64 different possible values for x_i. Complicating matters is that there isn't always a DPA peak for the correct secret index because the device leaks only for 16 consecutive guesses. For example, if $x_i = 2030$, then $y_i = \lfloor \frac{2030-1}{64} \rfloor \cdot 64 + 1 = 1985$ and the device will show leaks only for the 16 consecutive guesses from (1,985 to 2,000). Fortunately, more information is available if we look at the times at which the leaks occur.

We observed that the leak corresponding to y_i can appear in one of 8 different time locations corresponding to the 8-bit AVR memory-store operations. These 8 positions can be seen in Fig. 4. The upper plot shows the DPA results for the indices 1,985 to 1,992 (drawn in black) and other index values from 0 to 1,984 (drawn in gray). The lower plot shows the results for the indices 1,993 to 2,000, and other index values from 2,001 to 4,800. The leakage occurs during two 8-bit AVR memory-store operations near sample points 146 and 172. We discovered that the time location at which the leak for guess y_i occurs gives us more information about the correct value x_i.

Let $q_i \in \{0, ..., 7\}$ denote the location at which the leak corresponding to guess y_i occurs. It turns out that q_i is related to x_i by Eq. 17:

$$q_i = 7 - \lfloor \frac{(x_i - 1) \bmod 64}{8} \rfloor \in \{0, ..., 7\}. \tag{17}$$

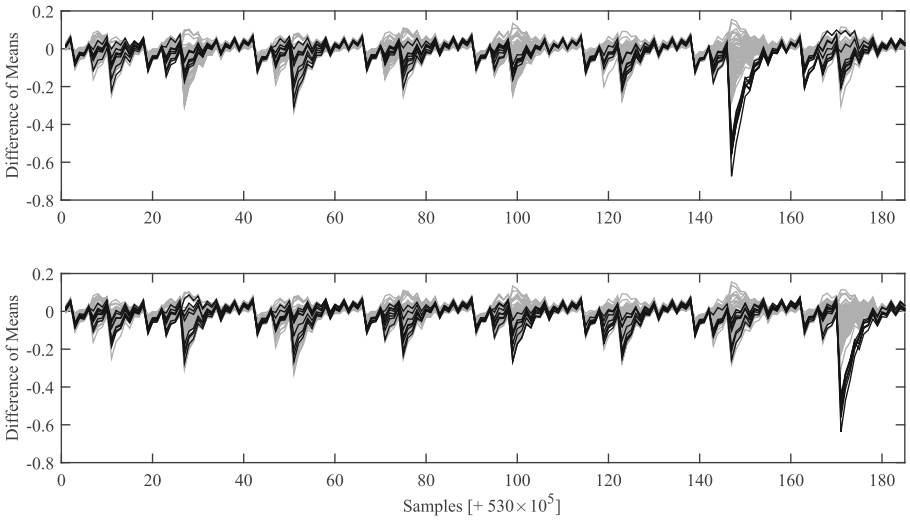

Fig. 4. Upper plot shows the DPA result using indices from 1,985 to 1,992 (drawn in black), the lower plot shows the result using indices from 1,993 to 2,000 (drawn in black). Other indices are drawn in gray.

In our example, $q_i = 7 - \lfloor \frac{2000-1 \bmod 64}{8} \rfloor = 6^{th}$ position. In Fig. 4, we see that the leak corresponding to $y_i = 1985$, in the upper plot, is in the 6th location.

Hence, using power analysis we were able to recover a pair of values (y_i, q_i) which narrows down the choice of x_i to one of 8 possible values. Given (y_i, q_i), there are only 8 possible values for x_i which satisfy both Eqs. 16 and 17:

$$x_i \in Z_i = [y_i + (7 - q_i) \times 8,\ y_i + (7 - q_i) \times 8 + 7]. \tag{18}$$

In our example we measured $(y_i, q_i) = (1985, 6)$, and therefore deduce that $Z_i = [1993, 2000]$.

3.4 About the Index Search Intervals Z_i

We denote by α the length of index search intervals Z_i. In a sense, α represents the precision of the DPA analysis. Our attack gave us search intervals of length $\alpha = 8$, which actually equals to the word width of the underlying AVR architecture. We assume that on other devices, with different architectures and word lengths, our attack could yield search intervals with different lengths. For example, on a 64-bit device, the search interval could have length $\alpha = 64$. We will see in Sect. 4 that the algebraic part of the attack is not feasible for such a large value of α. In this case, we recommend looking for ways to improve the precision of the power analysis step to reduce the size of the search intervals, or using a stronger method than we did for solving the noisy system of equations.

It may be the case that different secret indices lie in the same interval Z_i. We denote by β the total number of unique search intervals Z_i. Note that β satisfies

$\beta \leq \frac{w}{2}$. In our experiments, we needed around 100–200 traces to identify all β intervals of size $\alpha = 8$ containing the nonzero elements of $\boldsymbol{h}_0$. Figure 5 illustrates the intervals recovered.

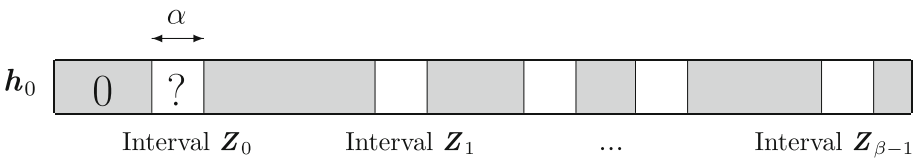

Fig. 5. Partial knowledge of $\boldsymbol{h}_0$ after the DPA attack.

4 Recovering the Rest of the Key

In this section we describe how we used the partial information discovered by our DPA attack to recover the rest of the key. A brute force attack could take up to $\alpha^{\frac{w}{2}}$ calculations, which would be infeasible. However, the sparseness of the private key enables a much more efficient attack.

We simply choose a large number of private key bit positions at random, and hope that all the bits in those positions are 0. Since over 99% of the private key bits are 0, our guess will be correct with non-negligible probability. Combined with the information recovered in the DPA attack, this will give us enough linear equations to solve for the private key. A more sophisticated attack might work with less information recovered, but our attack is sufficient for α up to 32.

4.1 Cryptanalytic Attack Using Partial Information of Secret Key

Recall that the public key is $\boldsymbol{P} = (\boldsymbol{H}_1^{-1} \cdot \boldsymbol{H}_0)^T$. Setting $\boldsymbol{Q} = \boldsymbol{P}^{-1}$ we rearrange and write

$$\boldsymbol{Q} \cdot \boldsymbol{H}_0^T = \boldsymbol{H}_1^T. \tag{19}$$

The matrices $\boldsymbol{H}_0$ and $\boldsymbol{H}_1$ are sparse circulants defined by their first rows $\boldsymbol{h}_0$ and $\boldsymbol{h}_1$ respectively. We can therefore write 19 as the system of linear equations

$$\boldsymbol{Q} \cdot \boldsymbol{h}_0^T = \boldsymbol{h}_1^T \tag{20}$$

where $\boldsymbol{Q}$ is dense and known, $\boldsymbol{h}_0$ is sparse and partially known as shown in Fig. 5, and $\boldsymbol{h}_1$ is sparse and unknown.

We now use the information we recovered about $\boldsymbol{h}_0$ to help us completely solve the system of Eq. in 20. First, we know the β intervals $\{\boldsymbol{Z}_0, ..., \boldsymbol{Z}_{\beta-1}\}$ of length α which contain all the nonzero entries of $\boldsymbol{h}_0$. All the entries of $\boldsymbol{h}_0$ outside these intervals are known to be zero. We can therefore remove from our system of equations the zero-valued entries of $\boldsymbol{h}_0$, and the corresponding columns of $\boldsymbol{Q}$. This leaves us with a new system of equations

$$\boldsymbol{Q}' \cdot \boldsymbol{h}_0'^T = \boldsymbol{h}_1^T \tag{21}$$

where $\boldsymbol{h}'_0 = (\boldsymbol{Z}_0, ..., \boldsymbol{Z}_{\beta-1})$ is the vector of length $\alpha\beta$ obtained by concatenating the variables in the intervals containing the nonzero entries of $\boldsymbol{h}_0$, and $\boldsymbol{Q'}$ is the $r \times \alpha\beta$ matrix obtained by removing from $\boldsymbol{Q}$ the columns corresponding to the zero-valued entries of $\boldsymbol{h}_0$. This step is illustrated in Fig. 6 below. We use the color gray to represent the removed variables.

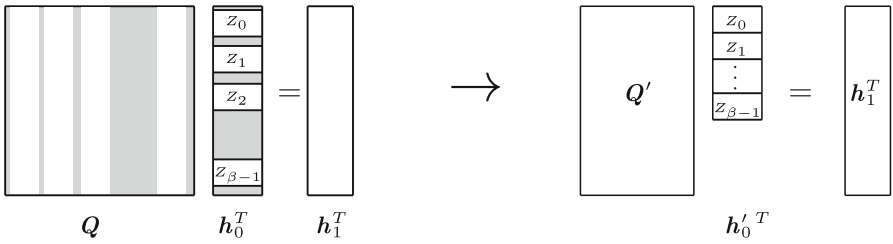

Fig. 6. Removing the columns of Q

The DPA attack allows us to know if two or more secret indices lie in the same interval Z_i. We therefore know the number of nonzero values of each interval of $\boldsymbol{h}_0$ and use this information to add parity equations to the system. Let b_i denote the number of nonzero values of the interval $\boldsymbol{Z}_i$ modulo 2. Then

$$b_i = (1, 1, ..., 1) \cdot \boldsymbol{Z}_i^T. \tag{22}$$

There will be exactly β such equations. Let $\boldsymbol{b} = (b_0, ..., b_{\beta-1})$ and $\boldsymbol{W}$ be the $\beta \times \alpha\beta$ matrix which for row i, $0 \leq i < \beta$, has ones in positions j for $i \cdot \alpha \leq j < (i+1) \cdot \alpha$ and zeros elsewhere. We can then extend our system of equations to include the parity equations by appending $\boldsymbol{W}$ to the bottom of $\boldsymbol{Q'}$ and appending $\boldsymbol{b}$ to $\boldsymbol{h}_1$. The new extended $(r+\beta) \times \alpha\beta$ system of equations is shown in Fig. 7 below.

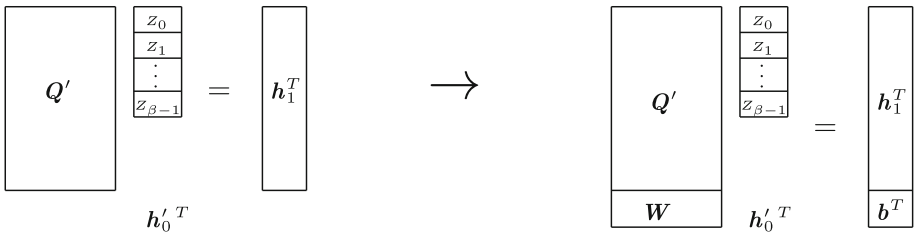

Fig. 7. Adding the parity equations

We don't know the vector $\boldsymbol{h}_1$. However, it is generated to be an extremely sparse vector and the entries are zero with probability $1 - \frac{w}{2r} > 0.99$. Suppose

we create a square $\alpha\beta \times \alpha\beta$ system of equations by randomly selecting $\beta(\alpha-1)$ entries from $\boldsymbol{h}_1$, and keeping the corresponding rows of $\boldsymbol{Q}'$. We also retain all the parity information $\boldsymbol{W}$ and $\boldsymbol{b}$. Then the probability p that all the randomly selected entries from $\boldsymbol{h}_1$ are zero is

$$p = \frac{\text{number of } \boldsymbol{h}_1 \text{ for which guess is right}}{\text{total possible number of } \boldsymbol{h}_1} \tag{23}$$

$$= \frac{\binom{r-\beta(\alpha-1)}{\frac{w}{2}}}{\binom{r}{\frac{w}{2}}} = \frac{(r-\beta(\alpha-1))!(r-\frac{w}{2})!}{r!(r-\beta(\alpha-1)-\frac{w}{2})!} \tag{24}$$

The expected number of attempts before finding a subvector of $\boldsymbol{h}_1$ with all zeros entries is $\frac{1}{p}$. Table 2 gives an estimation of this, using the parameters proposed for QcBits and assuming the worst case of $\beta = \frac{w}{2}$.

Table 2. Approximate number of attempts in the worst case

α =	8	16	32	64
80-bit	22	950	2^{23}	2^{58}
128-bit	40	3500	2^{26}	2^{64}

The last step in the attack proceeds as follows. We randomly select $\beta(\alpha-1)$ entries of $\boldsymbol{h}_1$, and guess that they are all zero. We also extract the corresponding rows of $\boldsymbol{Q}'$ and denote the resulting matrix $\boldsymbol{Q}''$. We retain all the parity information $\boldsymbol{W}$ and $\boldsymbol{b}$ as well, giving us a square $\alpha\beta \times \alpha\beta$ system of equations. This process is shown in Fig. 8 below. Here the color gray represents the rows that we keep.

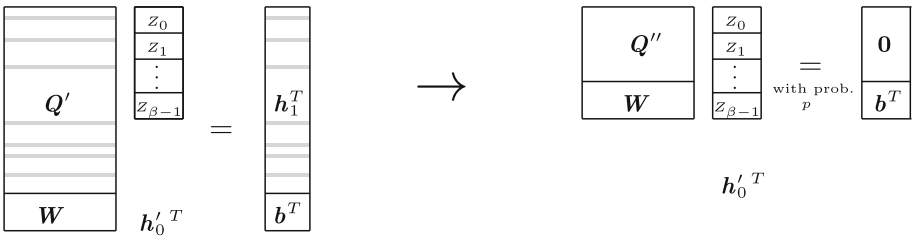

Fig. 8. Selecting random positions in $\boldsymbol{h}_1$ and corresponding rows of $\boldsymbol{Q}'$

Finally, we solve the system of equations

$$\begin{pmatrix} \boldsymbol{Q}'' \\ \boldsymbol{W} \end{pmatrix} \cdot \boldsymbol{h}_0'^{\,T} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{b}^T \end{pmatrix} \tag{25}$$

If all the selected entries of $\boldsymbol{h}_1$ are actually zero, then the correct value of $\boldsymbol{h}_0'$ is among the solutions. We then look for a solution vector ${\boldsymbol{h}_0'}^T$ with weight exactly $\frac{w}{2}$, and we also check that $\boldsymbol{Q} \cdot \boldsymbol{h}_0^T$ has weight exactly $\frac{w}{2}$. If this is the case, we have found $\boldsymbol{h}_0$, and $\boldsymbol{h}_1$ can be computed directly from it. If this is not the case, the selected entries of $\boldsymbol{h}_1$ are not all zero and a suitable solution will not be found. We then keep repeating the final step with different random subvectors of $\boldsymbol{h}_1$ until a solution is found.

4.2 Attack Complexity

To compute the attack's complexity, we include the cost of repeatedly solving $\alpha\beta \times \alpha\beta$ systems of binary linear equations. For our estimates, we assume the worst case, in which $\beta = \frac{w\alpha}{2}$. As for solving the system, Vassilevska Williams has an algorithm which can solve such a system with complexity $(\frac{w\alpha}{2})^{2.373}$ [44]. Hence the average total complexity of the algebraic part of our attack is

$$\frac{1}{p} \cdot \left(\frac{w\alpha}{2}\right)^{2.373} \tag{26}$$

In our experiments, the DPA attack gave us $\alpha = 8$. Hence, the total average complexity of our key recovery attack is 2^{24} for the 80-bit security version, and 2^{27} for the 128-bit security version.

4.3 Experimental Results

We verified the algebraic part of our attack using SAGE on one core of a 2.9 GHz Core i5 MacBook Pro. We tested the attack for $\alpha \in \{8, 16, 32\}$. For $\alpha \in \{8, 16\}$ we had a 100% success rate with a bounded number of iterations. We successfully recovered the secret key in each test, with at most 10,000 iterations. For $\alpha = 32$ with 80 bits of security, the expected time in the worst case of $\beta = \frac{w}{2}$ is around 16 h. For $\alpha = 32$ with 128 bits of security, and $\alpha = 64$, we estimated the expected times based on our experiments with the other α values.

The results are shown in Table 3, and the times shown exclude the preparation step of computing the initial matrix $\boldsymbol{Q}'$. Since the main loop of the attack is based on guessing subsets of the equations until a guess is correct, it is completely parallelizable. Thus the results should scale inversely with the number of cores used to perform the attack.

Table 3. Approximate solving times in SAGE on one core

α =	8	16	32	64
80 bits	0.4 s	15 s	16 h	≈530 years
128 bits	2 s	4 min	≈ 7 days	≈790,000 years

5 Attack Countermeasure

We propose a simple masking technique to help defend against side channel attacks during the syndrome calculation in QcBits. Since QC-MDPC codes are linear, the XOR of two codewords is another codeword. Also, all codewords are in the nullspace of the parity check matrix $\boldsymbol{H}_{priv}$. We can therefore mask the corrupted codeword $(\boldsymbol{c}|\boldsymbol{0})$ by XORing it with a random codeword $\boldsymbol{c}_m$ before passing it to the syndrome calculation. This does not change the outcome of the syndrome calculation since

$$\boldsymbol{H}_{priv} \cdot ((\boldsymbol{c}|\boldsymbol{0}) \oplus \boldsymbol{c}_m)^T = \boldsymbol{H}_{priv} \cdot (\boldsymbol{c}|\boldsymbol{0})^T \oplus \boldsymbol{H}_{priv} \cdot \boldsymbol{c}_m^T = \boldsymbol{H}_{priv} \cdot (\boldsymbol{c}|\boldsymbol{0})^T. \quad (27)$$

It does effectively mask the DPA leak we exploited, however. Figure 9 shows the difference of means for all possible guesses for x_i with this countermeasure implemented. In contrast to Fig. 3, there is no significant spike for any of the guesses.

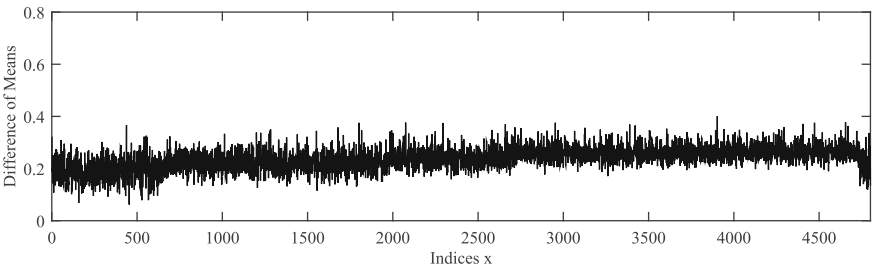

Fig. 9. Maximum Difference of Means (DoM) using 500 traces over all possible values x_i when the countermeasure is enabled. The right key index is 2000.

This countermeasure is of course only effective during the syndrome calculation. Additional side-channel countermeasures would be required to protect the private key during other calculations such as the bit flipping algorithm.

6 Conclusions

In this paper we described a key recovery attack against QcBits. We first performed power analysis to recover partial information about the key. We then used that information to set up and solve a system of noisy binary linear equations. Solving that system recovered the entire key. Finally, we proposed a simple countermeasure which was effective against the power analysis we performed in the attack.

QcBits has sparse, highly structured private keys. The sparseness is required for the decoding algorithm to work. The quasi-circulant nature of the keys is essential for small key sizes and efficient calculations. We exploited both these

features in our attack. Another characteristic of QcBits and other code-based algorithms is that the Hamming weight of the noise added to codewords during encryption must be modest enough that the corrupted word can be decoded.

Many proposals for post-quantum cryptography are based on noisy linear systems: lattices, learning with errors or error-correcting codes. In terms of side-channel resilience, these systems have an important difference from systems based on number-theoretic problems. Leaking a few bits of a number-theoretic system may open up a new avenue of attack, but it usually doesn't directly contribute to solving the underlying hard problem. For noisy linear systems, leaking a few bits of the secret is likely to directly erode the difficulty of the underlying hard problem. Therefore designers and analysts may wish to consider the risks of side-channel analysis when evaluating post-quantum cryptographic algorithms.

References

1. Augot, D., Batina, L., Bernstein, D.J., Bos, J., Buchmann, J., Castryck, W., Dunkelman, O., Güneysu, T., Gueron, S., Hülsing, A. , Lange, T., Emam Mohamed, M.S., Rechberger, C., Schwabe, P., Sendrier, N., Vercauteren, F., Yang, B.-Y.: Initial recommendations of long-term secure post-quantum systems (2015). http://pqcrypto.eu.org/docs/initial-recommendations.pdf. 4, 5
2. Belaïd, S., Coron, J.-S., Fouque, P.-A., Gérard, B., Kammerer, J.-G., Prouff, E.: Improved side-channel analysis of finite-field multiplication. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 395–415. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48324-4_20. 6
3. Belaïd, S., Fouque, P.-A., Gérard, B.: Side-channel analysis of multiplications in GF(2^{128}) - application to AES-GCM. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 306–325. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45608-8_17. 6
4. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 26–45. Springer, Heidelberg (1998). doi:10.1007/BFb0055718. 5
5. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.: On the inherent intractability of certain coding problems. IEEE Trans. Inf. Theory **24**(3), 384–386 (1978). 7
6. Bernstein, D.J.: The Poly1305-AES message-authentication code. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 32–49. Springer, Heidelberg (2005). doi:10.1007/11502760_3. 8
7. Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.): Post-Quantum Cryptography. Springer, Heidelberg (2009). 4
8. Bernstein, D.J., Schwabe, P.: New AES software speed records. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 322–336. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89754-5_25. 8
9. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. In: Frances Yao, F., Luks, E.M. (eds.) Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, Portland, OR, USA, 21–23 May 2000, pp. 435–440. ACM (2000). 6
10. Chaulet, J., Sendrier, N.: Worst case QC-MDPC decoder for McEliece cryptosystem. In: IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, 10–15 July 2016, pp. 1366–1370. IEEE (2016). 5

11. Chen, C., Eisenbarth, T., Maurich, I., Steinwandt, R.: Differential power analysis of a McEliece cryptosystem. In: Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M. (eds.) ACNS 2015. LNCS, vol. 9092, pp. 538–556. Springer, Cham (2015). doi:10.1007/978-3-319-28166-7_26. 5
12. Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on post-quantum cryptography. National Institute of Standards and Technology (NIST), NISTIR 8105 Draft, U.S. Department of Commerce, February 2016. 4
13. Chou, T.: QcBits: constant-time small-key code-based cryptography. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 280–300. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53140-2_14. 4, 5, 6, 8
14. Couvreur, A., Corbella, I.M., Pellikaan, R.: A polynomial time attack against algebraic geometry code based public key cryptosystems. In: 2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, 29 June–4 July 2014, pp. 1446–1450. IEEE (2014). 4
15. Gallager, R.G.: Low-density parity-check codes. IRE Trans. Information Theory **8**(1), 21–28 (1962). 7
16. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. Cryptology ePrint Archive, Report 2016/858 (2016). http://eprint.iacr.org/2016/858. 5, 6
17. Heyse, S., Moradi, A., Paar, C.: Practical power analysis attacks on software implementations of McEliece. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 108–125. Springer, Heidelberg (2010). doi:10.1007/978-3-642-12929-2_9. 5
18. Heyse, S., Maurich, I., Güneysu, T.: Smaller keys for code-based cryptography: QC-MDPC McEliece implementations on embedded devices. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 273–292. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40349-1_16. 4
19. Janwa, H., Moreno, O.: McEliece public key cryptosystems using algebraic-geometric codes. Des. Codes Crypt. **8**(3), 293–307 (1996). 4
20. Kobara, K., Imai, H.: Semantically secure McEliece public-key cryptosystems - conversions for McEliece PKC. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 19–35. Springer, Heidelberg (2001). doi:10.1007/3-540-44586-2_2. 5
21. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_25. 5
22. Landais, G., Tillich, J.-P.: An efficient attack of a McEliece cryptosystem variant based on convolutional codes. Cryptology ePrint Archive, Report 2013/080 (2013). http://eprint.iacr.org/2013/080. 4
23. Löndahl, C., Johansson, T.: A new version of McEliece PKC based on convolutional codes. In: Chim, T.W., Yuen, T.H. (eds.) ICICS 2012. LNCS, vol. 7618, pp. 461–470. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34129-8_45. 4
24. McEliece, R.J.: A public-key system based on algebraic coding theory. DSN Progress Report 44, pp. 114–116 (1978). 4
25. Minder, L., Shokrollahi, A.: Cryptanalysis of the Sidelnikov cryptosystem. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 347–360. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72540-4_20. 4
26. 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, Istanbul, Turkey, 7–12 July 2013, pp. 2069–2073. IEEE (2013). 4, 5, 7

27. Monico, C., Rosenthal, J., Shokrollahi, A.: Using low density parity check codes in the McEliece cryptosystem. In: IEEE International Symposium on Information Theory, ISIT 2000, p. 215 (2000). 4
28. Mosca, M.: Cybersecurity in an era with quantum computers: will we be ready? Cryptology ePrint Archive, Report 2015/1075 (2015). http://eprint.iacr.org/2015/1075. 4
29. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. Probl. Control Inf. Theory **15**, 159–166 (1986). 4, 8
30. O'Flynn, C., Chen, Z.D.: ChipWhisperer: an open-source platform for hardware embedded security research. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 243–260. Springer, Cham (2014). doi:10.1007/978-3-319-10175-0_17. 11
31. Peeters, M., Van Assche, G., Bertoni, G., Daemen, J.: Keccak and the SHA-3 standardization (2013). http://csrc.nist.gov/groups/ST/hash/sha-3/documents/Keccak-slides-at-NIST.pdf. 8
32. Persichetti, E.: Secure and anonymous hybrid encryption from coding theory. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 174–187. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38616-9_12. 5
33. Schneier, B.: NSA plans for a post-quantum world (2015). https://www.schneier.com/blog/archives/2015/08/nsa_plans_for_a.html. 4
34. Sendrier, N.: On the concatenated structure of a linear code. Appl. Algebra Eng. Commun. Comput. **9**(3), 221–242 (1998). 4
35. Seurin, Y.: Primitives et protocoles cryptographiques à sécurité prouvée. Ph.D. thesis, Université de Versailles Saint-Quentin-en-Yvelines (2009). Sect. 3.5.7. 6
36. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings of the 35th FOCS, pp. 124–134. IEEE Computer Society Press, November 1994. 3
37. Shoufan, A., Strenzke, F., Molter, H.G., Stöttinger, M.: A timing attack against patterson algorithm in the McEliece PKC. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 161–175. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14423-3_12. 5
38. Sidelnikov, V.M.: A public-key cryptosystem based on binary Reed-Muller codes. Discret. Math. Appl. **4**(3), 191–208 (1994). 4
39. Sidelnikov, V.M., Shestakov, S.O.: On insecurity of cryptosystems based on generalized Reed-Solomon codes. Discret. Math. Appl. **2**(4), 439–444 (1992). 4
40. Strenzke, F.: A timing attack against the secret permutation in the McEliece PKC. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 95–107. Springer, Heidelberg (2010). doi:10.1007/978-3-642-12929-2_8. 5
41. Strenzke, F., Tews, E., Molter, H.G., Overbeck, R., Shoufan, A.: Side channels in the McEliece PKC. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 216–229. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88403-3_15. 5
42. von Maurich, I., Güneysu, T.: Lightweight code-based cryptography: QC-MDPC McEliece encryption on reconfigurable devices. In: Fettweis, G., Nebel, W. (eds.) Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, 24–28 March 2014, pp.1–6. European Design and Automation Association (2014). 4

43. Maurich, I., Heberle, L., Güneysu, T.: IND-CCA secure hybrid encryption from QC-MDPC Niederreiter. In: Takagi, T. (ed.) PQCrypto 2016. LNCS, vol. 9606, pp. 1–17. Springer, Cham (2016). doi:10.1007/978-3-319-29360-8_1. 5
44. Williams, V.V.: Multiplying matrices faster than Coppersmith-Winograd. In: Karloff, H.J., Pitassi, T. (eds.) Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, 19–22 May 2012, pp. 887–898. ACM (2012). 18