Protecting HQC decryption from side-channel differential power analysis

Domenico Cacace – 977835

Abstract

This work describes a masking-based side channel resistent implementation of HQC, a third round alternative candidate for the NIST Post-Quantum Cryptography competition. The presented implementation is then tested on an ARM Cortex-M4 processor; the test results show a significant decrease in terms of leaked information, while keeping the overhead factor due to the masking below 2.

I. Introduction

The security of current public key cryptosystems is based on the *hardness* of some mathematical problems, such as large integer factorization or discrete logarithm extraction; however, such problems could become tractable, thanks to the Shor's algorithm, with the aid of a large scale quantum computer. For this reason in 2017 the National Institute of Standards and Technology (NIST) started a procedure for the standardization of cryptographic primitives able to withstand such kind of attacks.

Many proposals, such as Hamming Quasi-Cyclic (HQC), are based on *hard* problems of coding theory; even though, the implementation of the cryptosystem can be vulerable to side channel attacks, that can exploit the information leaked by phisically measurable channels (*e.g.* EM fields, power consumption).

In this paper we make a brief recap of the basics of coding theory and an overview of the cryptosystem. We present a masking scheme suitable for HQC, then implement and benchmark this scheme on an ARM Cortex-M4 microprocessor.

II. Preliminaries

General definitions

We denote with $\mathbb{F}_2$ the binary finite field, with $\mathbb{Z}$ the integer ring and with $\mathcal{V} = \mathbb{F}_2^n$ a vector space of dimension n over $\mathbb{F}_2$ for some positive integer $n \in \mathbb{Z}$; elements of $\mathcal{V}$ can be interchangeably considered as row vectors or polynomials in $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$. Additionally, we denote by $\omega(\cdot)$ the Hamming weight of a vector *i.e.* the number of its non-zero coordinates.

Let $\mathbf{v} = (v_0, \cdots, v_{n-1}) \in \mathcal{V}$. The circulant matrix induced by v is defined as

$$\mathbf{rot}(\mathbf{v}) = \begin{pmatrix} v_0 & v_{n-1} & \cdots & v_1 \\ v_1 & v_0 & \cdots & v_2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{n-1} & v_{n-2} & \cdots & v_0 \end{pmatrix} \in \mathbb{F}_2^{n \times n}$$

Coding theory[1]

A linear code $\mathcal{C}$ of length n and dimension k (denoted as $[n, k]$) is a subspace of $\mathcal{V}$ of dimension k. Elements of $\mathcal{C}$ are called *codewords*.

We say that $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ is a generator matrix for the code $\mathcal{C}$ if

$$\mathcal{C} = \{\mathbf{mG}, \text{for } \mathbf{m} \in \mathbb{F}_2^k\}$$

Given a $[n, k]$ code $\mathcal{C}$, we say that the matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ is a Parity-Check matrix for $\mathcal{C}$ if it is a generator matrix of the dual code $\mathcal{C}^\perp$ or, in other words,

$$\mathcal{C} = \{\mathbf{v} \in \mathbb{F}_2^n \text{ such that } \mathbf{Hv}^\top = \mathbf{0}\}$$

Given $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ a parity matrix for some code $\mathcal{C}$ and a word $\mathbf{v} \in \mathbb{F}_2^n$, we call $\mathbf{Hv}^\top$ the syndrome of $\mathbf{v}$. From the definition of parity-check matrix, it holds that:

$$\mathbf{v} \in \mathcal{C} \iff \mathbf{Hv}^\top = \mathbf{0}$$

Given a linear code $\mathcal{C}$ over $\mathcal{V}$ and ω a norm on $\mathcal{V}$, we define the minimum distance of $\mathcal{C}$ as

$$d = \min_{\mathbf{u}, \mathbf{v} \in \mathcal{C}, \mathbf{u} \neq \mathbf{v}} \omega(\mathbf{u} - \mathbf{v})$$

A code with minimum distance d is capable of decoding an arbitrary pattern of up to $\delta = \lfloor \frac{d-1}{2} \rfloor$ errors; such a code is also denoted as $[n, k.d]$.

Consider a vector $\mathbf{c} = (\mathbf{c}_0, \cdots, \mathbf{c}_{s-1}) \in \mathbb{F}_2^{sn}$ as s successive n-uples; a $[sn, k, d]$ linear code $\mathcal{C}$ is said to be Quasi-Cyclic (QC) of order s if, for any $\mathbf{c} = (\mathbf{c}_0, \cdots, \mathbf{c}_{s-1}) \in \mathcal{C}$, the vector obtained by applying a simultaneous circular shift to every block $\mathbf{c}_i$ is also a codeword.

Decoding

When dealing with linear codes, the problem of decoding a vector stays the same when we use the syndrome of the vector, so we speak of Syndrome Decoding (SD).

For positive integers n, k and w the $\mathrm{SD}(n, k, w)$ distribution *chooses* $\mathbf{H} \xleftarrow{\$} \mathbb{F}_2^{(n-k)\times n}$ and $\mathbf{x} \xleftarrow{\$} \mathbb{F}_2^n$ such that $\omega(\mathbf{x}) = w$ and outputs the couple $(\mathbf{H}, \mathbf{H}\mathbf{x}^\top)$.

On input $(\mathbf{H}, \mathbf{y}^\top)$ from the SD Distribution, the Syndrome Decoding problem $\mathrm{SD}(n, k, w)$ asks to find a vector $\mathbf{x}$ such that $\mathbf{H}\mathbf{x}^\top = \mathbf{y}^\top$ and $\omega(\mathbf{x}) = w$; for the Hamming distance this problem has been proven to be NP-complete [2].

The decisional version of the SDP requires to determine, given $(\mathbf{H}, \mathbf{y}^\top) \in \mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{(n-k)}$, if $(\mathbf{H}, \mathbf{y}^\top)$ comes from the $\mathrm{SD}(n, k, w)$ or the uniform distribution over $\mathbb{F}_2^{(n-k)\times n} \times \mathbb{F}_2^{(n-k)}$; in other words, this is the problem of decoding random linear codes from random errors.

III. Cryptosystem Overview

Hamming Quasi-Cyclic (HQC) is a cryptosystem running for standardization to NIST's competition in the category "post-quantum public key encryption scheme" [3]. It is a code-based cryptosystem, and its security is based on the hardness of a variant of the Decisional SD Problem, reported in [4].
In order to cope with the huge key sizes associated with code-based cryptosystems, HQC adopts Quasi-Cyclic (QC) codes [5]; there is no general complexity result for the decoding of QC codes, but this problem is considered hard by the community.

HQC consists of two schemes: a Public Key Encryption (PKE) and a Key Encapsulation/Data Encapsulation Mechanism (KEM/DEM).

HQC.PKE

The PKE employs a decodable $[n, k]$ code $\mathcal{C}$ generated by $\mathbf{G} \in \mathbb{F}_2^{k\times n}$, which can correct at most δ errors with the $\mathcal{C}.\mathsf{Decode}(\cdot)$ algorithm, and a random double-circulant $[2n, n]$ code with parity-check matrix $(\mathbf{1}, \mathbf{h})$; the algorithms that constitute this scheme are the following:

- $\mathsf{Setup}()$: generate and returns the parameters $\mathsf{param}=(n, k, \delta, w, w_\mathbf{r}, w_\mathbf{e})$.
- $\mathsf{KeyGen}(\mathsf{param})$: sample a random vector $\mathbf{h}$ for the parity-check matrix, the generator matrix $\mathbf{G}$ (which is public), the private key pk as a couple of random vectors $(\mathbf{x}, \mathbf{y}) \in \mathcal{R}^2$ both oh Hamming weight equal to w, then set the public key $\mathsf{pk} = (\mathbf{h}, \mathbf{s} \leftarrow \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$; return the keys $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m})$: generate the random vectors $\mathbf{e} \in \mathcal{R}$, $(\mathbf{r}_1, \mathbf{r}_2) \in \mathcal{R}^2$ such that $\omega(\mathbf{e}) = w_\mathbf{e}$ and $\omega(\mathbf{r}_1) = \omega(\mathbf{r}_2) = w_\mathbf{r}$, then calculate the ciphertext as $\mathbf{c} = (\mathbf{u}, \mathbf{v}) = (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2, \mathbf{m}\mathbf{G} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e})$
- $\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$: return $\mathcal{C}.\mathsf{Decode}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$

Although sk is composed of two vectors, $\mathbf{x}$ is only used to compute part of pk and it is not employed in the decryption phase.

HQC.KEM

The KEM is built upon an instance of $\mathsf{HQC.PKE}$ $\mathcal{E}$ and three hash functions $\mathcal{G}, \mathcal{H}, \mathcal{K}$, with $\mathcal{G} \neq \mathcal{H}$. The setup and key generation algorithms are the same of the PKE, with the exception that k now is the lenght of the shared key.

- $\mathsf{Encaps}(\mathsf{pk})$: generate a random $\mathbf{m} \in \mathbb{F}_2^k$ to derive the randomness $\theta \leftarrow \mathcal{G}(\mathbf{m})$; generate the ciphertext $\mathbf{c} \leftarrow \mathcal{E}.\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$, derive the symmetric key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$ and send $(\mathbf{c}, \mathbf{d} = \mathcal{H}(\mathbf{m}))$
- $\mathsf{Decaps}(\mathsf{sk}, \mathbf{c}, \mathbf{d})$: decrypt $\bar{\mathbf{m}} \leftarrow \mathcal{E}.\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$, compute $\bar{\theta} \leftarrow \mathcal{G}(\bar{\mathbf{m}})$ and use them to calculate $\bar{\mathbf{c}} \leftarrow \mathcal{E}(\mathsf{pk}, \bar{\mathbf{m}}, \bar{\theta})$: at this point if both $\mathbf{c} = \bar{\mathbf{c}}$ and $\mathbf{d} = \mathcal{H}(\bar{\mathbf{m}})$ we can derive the shared key $K \leftarrow \mathcal{K}(\bar{\mathbf{m}}, \bar{\mathbf{c}})$.

IV. Masking

An approach used to design countermeasures against side channel attacks consists in applying *secret sharing schemes* [6]. In this scheme, the sensitive data is randomly split into a number of shares, in such a way that to reconstruct any information about the data, a chosen number (*threshold*) of shares are needed. In particular, as proven in [7], a *d-th* order masking scheme (data

split into $d+1$ shares) with a threshold of $d+1$ is a sound countermeasure against SCA in a realistic leakage model([8]).
The basic principle of masking to secure a cryptographic algorithm is to randomly split every sensitive variable x into $d+1$ shares $x_0, \cdots, x_d$, in such a way that the following relation holds:

$$x_0 \oplus x_1 \oplus \cdots \oplus x_d = x$$

Ususally the shares $x_1, \cdots, x_d$, also called *masks*, are picked at random, while the *masked variable* x_0 is processed in such a way that it satisfies the relation above. If the d masks are uniformly distributed, masking makes every intermediate variable statistically independent from the secret, thus disallowing a *d-th* order SCA. A *(d+1)-th* SCA could still be feasible, however such attacks become impractical as d increases; as a consequence, the masking order is a security parameter of the cryptosystem.

Securing operations

It is fundamental, in order to protect the secrets, to implement mathematical operations in such a way that no valuable information is leaked through some side channel; in [8] Ishai, Sahai and Wagner proposed an algorithm to securely compute a logic AND gate for any number of shares.

Algorithm 1: ISW Multiplication

Input: sharings $(a_1, \cdots, a_d)$, $(b_1, \cdots, b_d) \in \mathbb{F}_{2^n}$
Output: sharings $(c_1, \cdots, c_d)$ such that $\bigoplus_i c_i = (\bigoplus_i a_i) \wedge (\bigoplus_i b_i)$

for $i \leftarrow 1$ **to** d **do**
  $c_i = a_i \wedge b_i$
end
for $i \leftarrow 1$ **to** d **do**
  for $j \leftarrow i+1$ **to** d **do**
    $s \xleftarrow{\$} \mathbb{F}_{2^n}$
    $s' \leftarrow (s \oplus (a_i \wedge b_j)) \oplus (a_j \wedge b_i)$
    $c_i \leftarrow c_i \oplus s$
    $c_j \leftarrow c_j \oplus s'$
  end
end
return $(c_1, \cdots, c_d)$

The algorithm above is a generalization of the original one, where shares are elements of the field $\mathbb{F}_{2^n}$ and the operations $\oplus$ and $\wedge$ are the bitwise XOR and AND operations on the same field.
This algorithm has been proven secure in [9] against the probing model and in [10] against the Strong Non-Interference model.

V. Implementation

As a starting point, we take the third version of the reference implementation[1]; this is a self-contained C implementation that can be ported on virtually any architecture as-is.

Target

Our target architecture is `ARMv7E`: in particular, our tests are run on the STM F401-RE board mounting a Cortex® M4 processor. This particular board is equipped with 512KB of flash memory and 96KB of RAM: due to this characteristics and the large memory footprint of the cryptosystem, we were able to run HQC only at security level 1 (corresponding to 128 bits of security). In order to interact with the system we employ the Hardware Abstraction Layer (HAL) drivers, generated by STM32CubeMX specifically for our configuration. We use the drivers to notify when the encryption process is taking place through the LED on the board and to allow communications over the USB debug interface, redirecting in the `syscalls.c` file the standard output to the debug bridge for logging purposes. To generate random variables required we employ the `seedexpander` function from SHAKE256 [11].

Objects representations

Elements of the various binary fields are represented as arrays. For randomly generated values, we employ the `seedexpander` function on a 40B seed.
The secret key $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$ is generated by **seed1**, while the parity matrix $\mathbf{h}$ of the public key $\mathsf{pk}=(\mathbf{h}, \mathbf{s})$ is generated by **seed2**; the same principle applies for the random masks employed in the `safe_mul` function. These values are sampled uniformly from the respective fields, with a given Hamming weight.
In all the multiplications, one of the operands is always a sparse polynomial: these polynomials are represented with a position vector of ω coordinates, containing the orders of the nonzero coefficients.

[1] https://pqc-hqc.org/download.php?file=hqc-reference-implementation_2021-06-06.zip

Multiplication and masking

The multiplication over $\mathbb{F}_2[X]/(X^n-1)$ is implemented with a slight variation of the scoolbook multiplication algorithm, since the sparsity of one of the operands yields a lower computational complexity than other methods. This approach leaks information about the operands, which should remain secret: to overcome this problem we implemented a first-order masked multiplication scheme.

The dense (a) and sparse (b) polynomials are split in two halves, which are then combined accordingly to the ISW scheme; the `add` and `mult` function are an abstraction of the instructions that perform the addition and multiplication.

Algorithm 2: First-order masked multiplication, 2 shares

Input: $a = (a_0 a_1) \in \mathbb{F}_2^n$ dense polynomial,
$b = (b_0 b_1)$ sparse polynomial, $\omega(b) = w$
Output: $res, mask$ such that $res \oplus mask = a \cdot b$
Data: $temp_1, temp_2 \in \mathbb{F}_2^n$

1 $mask \xleftarrow{\$_w} \mathbb{F}_2^n$
2 $temp_1 \leftarrow \texttt{mult}(a_1, b_0)$
3 $temp_2 \leftarrow \texttt{mult}(a_0, b_1)$
4 $res \leftarrow \texttt{add}(mask, temp_1)$
5 $res \leftarrow \texttt{add}(res, temp_2)$
6 $temp_1 \leftarrow \texttt{mult}(a_1, b_1)$
7 $temp_2 \leftarrow \texttt{mult}(a_0, b_0)$
8 $res \leftarrow \texttt{add}(res, temp_2)$
9 $mask \leftarrow \texttt{add}(mask, temp_1)$
10 **return** res, $mask$

We then generalized the masking procedure for any number of shares by creating a small code generator to unroll the loops and reduce the execution time; we implemented and tested the masking procedure for $d \in \{2, 3, 4\}$, as the number of operations grows with $\mathcal{O}(d^2)$.

VI. Results

Our implementation comes with some tools that allow us to test and benchmark the implementation. Each of these tools has been compiled with the flags `-mcpu=cortex-m4 -mthumb -ffunction-sections -fdata-sections -std=c99 -funroll-all-loops -pedantic -O3`, prioritizing the execution time more than the memory footprint.

Functional testing

Before testing the performance of the scheme, we need to verify that the cryptosystem works as intended. To check this, employ the same source code provided by the authors, checking that, when applying the KEM, the exchanged keys match.

Execution time

In our benchmark we measure the execution time of the primitives of both the KEM and PKE schemes, as well as the multiplication. To measure the number of clock cycles needed for each operation, we use the system timer. On a sample of 1000 runs, we get the following results:

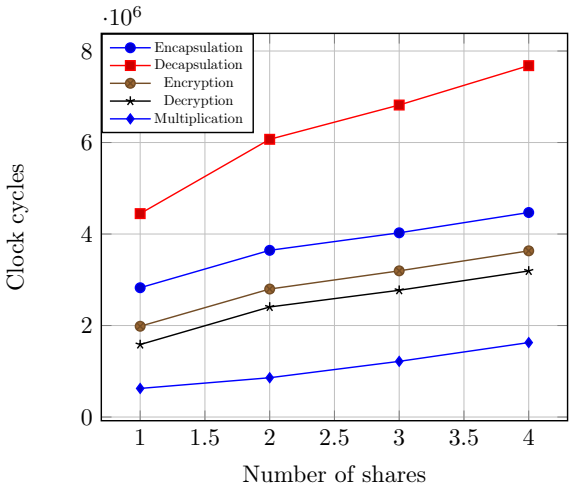

Figure 1: *Execution time (clock cycles)*

#	encaps	decaps	enc	dec	mul
2	22.55	29.88	31.57	43.62	33.58
3	35.43	45.91	50.27	65.44	43.48
4	50.35	64.32	70.83	90.65	91.88

Table 1: *Percentual performance decrease*

As we can observe, the overhead factor introduced by the masking operations is less than 2 for any operation, unlike the results obtained for some other cryptosystems ([12, 13, 14])

Constant-time execution

To detect an eventual leakage we rely on the *t-test*-based Test Vector Leakage Assessment (TVLA, [15]): we compare the measurements between two populations (fixed *vs* random) of 1000 samples each, and affirm that there is no correlation between the two if $|t| < 4.5$.

For both primitives we precompute keypair and secret message for the fixed measurements, while generating fresh randomness for the masks; we then measure the execution time as for the timing test, and obtain the following results:

#	encaps	decaps	enc	dec	mul
1	0.93666	4.56649	25.66178	10.26449	30.09420
2	6.00482	3.35650	4.62100	5.96089	4.96460
3	0.28901	1.69393	1.59911	5.88290	3.50812
4	0.12693	NA	3.30161	4.01299	0.21834

Table 2: *Welch's t (abs. value)*

We can note how the values tend to decrease for the KEM, PKE and multiplication, hinting that by applying masking mechanisms of increasing order the information leakage decreases. There are some anomalies in this trend that might be caused by a *bad* initialization of the PRNG; this should be investigated by repeating the tests on a similar board equipped with a TRNG.

VII. Conclusions

In this paper we presented the concept of masking to the HQC cryptosystem, in order to obfuscate the secret data during arithmetic operations and thus to reduce the likelihood of success of a side-channel attack.

Our experimental results show that the masked implementation of the multiplication algorithm reduces significantly the probability of success of a timing side-channel attack on all the major functions (encrypt, decrypt, encaps, decaps, multiplication), with a relatively small overhead.

As future work, it is possible to verify on a phisical board the resistance to DPA attacks. Furthermore, it is possible to port this implementation for the `stm32f4discovery` boards, to test the cryptosystem for higher security levels and to compare the results to the other candidates on the PQM4 project.

References

[1] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*, vol. 16. Elsevier, 1977.

[2] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[3] C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, G. Zémor, and I. Bourges, "Hamming quasi-cyclic (hqc)," *NIST PQC Round*, vol. 2, pp. 4–13, 2018.

[4] C. Aguilar-Melchor, O. Blazy, J.-C. Deneuville, P. Gaborit, and G. Zémor, "Efficient encryption from random quasi-cyclic codes," *IEEE Transactions on Information Theory*, vol. 64, no. 5, pp. 3927–3943, 2018.

[5] P. Gaborit, "Shorter keys for code based cryptography," in *Proceedings of the 2005 International Workshop on Coding and Cryptography (WCC 2005)*, pp. 81–91, 2005.

[6] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, p. 612–613, Nov. 1979.

[7] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Annual International Cryptology Conference*, pp. 398–412, Springer, 1999.

[8] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," in *Annual International Cryptology Conference*, pp. 463–481, Springer, 2003.

[9] M. Rivain and E. Prouff, "Provably secure higher-order masking of aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 413–427, Springer, 2010.

[10] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, P.-Y. Strub, and R. Zucchini, "Strong non-interference and type-directed higher-order masking," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 116–129, 2016.

[11] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015.

[12] S. Belaïd, P.-E. Dagand, D. Mercadier, M. Rivain, and R. Wintersdorff, "Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations," in *Eurocrypt 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, vol. 12107 of *Lecture Notes in Computer Science*, (Zagreb / Virtual, Croatia), pp. 311–341, Springer, May 2020.

[13] M. V. Beirendonck, J.-P. D'Anvers, A. Karmakar, J. Balasch, and I. Verbauwhede, "A side-channel resistant implementation of saber." Cryptology ePrint Archive, Report 2020/733, 2020. `https://ia.cr/2020/733`.

[14] J. W. Bos, M. Gourjon, J. Renes, T. Schneider, and C. van Vredendaal, "Masking kyber: First- and higher-order implementations." Cryptology ePrint Archive, Report 2021/483, 2021. `https://ia.cr/2021/483`.

[15] T. Schneider and A. Moradi, "Leakage assessment methodology," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 495–513, Springer, 2015.