Protecting HQC against differential power analysis attacks

Cryptography and Architectures for Computer Security project

Domenico Cacace

December 12, 2021

Introduction

Quantum Computing 101

What is a quantum computer?

A device that performs computations by exploiting the properties of quantum states:

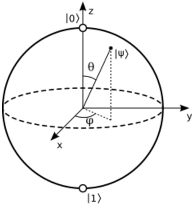
- **Superposition**: the combination of quantum states is a quantum state
- **Entanglement**: the quantum state of a particle depends on the state of other particles

Quantum Computing 101

What is a qubit?

- The basic unit of quantum information
- Represented as a quantum superposition of two basis states

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$



Classic computers *vs* classic cryptography

Symmetric cryptosystems

Key enumeration: perform an exhaustive search on the whole keyspace, $\mathcal{O}(n)$ time

Asymmetric cryptosystems

General number field sieve: factor a composite number n is $\mathcal{O}(e^{1.9(\log n)^{1/3}(\log \log n)^{2/3}})$ (subexponential) time

Quantum computers *vs* classic cryptography

Symmetric cryptosystems

Grover's algorithm: search in a database of n elements in $\mathcal{O}(n^{1/2})$ time and $\mathcal{O}(\log n)$ space

Asymmetric cryptosystems

Shor's algorithm: factor a composite number n, on a quantum computer, in $\mathcal{O}((\log n)^2(\log\log n)(\log\log\log n))$ (polynomial) time

NIST Post-Quantum Cryptography Standardization

Timeline

- 26/02/16: Announcement of NIST's call for submissions
- 21/12/17: Announcement of the first round candidates (69)
- 30/01/19: Announcement of the second round candidates (26)
- 22/07/20: Announcement of the third round candidates (7 finalists, 8 alternatives)
- 2022-2024: Publication of the standard drafts

NIST Post-Quantum Cryptography Standardization

Classes of cryptosystems

- **Lattice**: KYBER, FrodoKEM, NTRU Prime, SABER, ...
- **Code-based**: BIKE, Classic McElliece, HQC, LEDAcrypt, ...
- **Supersingular elliptic curve isogeny**: SIKE
- **Mutivariate**: CFPKM, Giophantus

We now focus on code-based cryptosystems.

Coding Theory 101

Codes

Let $\mathcal{V}$ be a vector space of dimension n over $\mathbb{F}_2$[1].

Linear codes

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

Quasi-cyclic codes

A **quasi-cyclic code** $\mathcal{C}$ is a linear code $[n, k]$ such that, for b (*index* of the code) coprime with n, if $c(x)$ is a codeword then $x^b c(x) \mod (x^n - 1)$ is also a codeword.

[1] $\mathcal{V}$ is isomorph to the polynomial ring $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$

Generating the code

Let $\mathcal{C}$ be a $[n, k]$ linear code.

Generator Matrix

A **generator matrix** of $\mathcal{C}$ is $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ if

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

Parity check Matrix

A **parity check matrix** of $\mathcal{C}$ is $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ if

$$\mathcal{C} = \{\mathbf{v} \in \mathbb{F}_2^k \mid \mathbf{Hv}^\top = \mathbf{0}\}$$

QC codes properties

Circulant matrix

Let $\mathbf{v} \in \mathbb{F}_2^n = (v_0, \cdots, v_{n-1})$. The circulant matrix induced by $\mathbf{v}$ is defined as:

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

A circulant matrix is entirely defined by any of its rows.

QC codes properties

QC codes parity-check matrix

Let $\mathcal{C}$ be a systematic quasi-cyclic code $[sn, n]$ of index s. The parity-check matrix of $\mathcal{C}$ is built as:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_n & 0 & \cdots & 0 & \mathbf{A}_1 \\ 0 & \mathbf{I}_n & & 0 & \mathbf{A}_2 \\ \vdots & & \ddots & 0 & \vdots \\ 0 & 0 & \cdots & \mathbf{I}_n & \mathbf{A}_{s-1} \end{bmatrix}$$

Where $\mathbf{A}_1, \cdots, \mathbf{A}_{s-1}$ are $n \times n$ circulant matrices.

Dual codes

Let $\mathcal{C}$ be a $[n, k]$ code, $\mathbf{G}$ and $\mathbf{H}$ its generator and parity check matrices, and $\mathbf{v}$ a codeword of $\mathcal{C}$

Dual code

$\mathbf{H}$ is the generator of the dual code $\mathcal{C}^\perp$

$$\mathbf{G} = \left[I_k \mid P \right] \qquad \mathbf{H} = \left[-P^\top \mid I_{n-k} \right]$$

Syndrome

We call **syndrome** of $\mathbf{v}$ the product $\mathbf{H}\mathbf{v}^\top$. From the definition, we have:

$$\mathbf{v} \in \mathcal{C} \Leftrightarrow \mathbf{H}\mathbf{v}^\top = \mathbf{0}$$

Metrics

Let $\mathcal{C}$ be a $[n, k]$ code, and $\mathbf{v}, \mathbf{w}$ codewords of $\mathcal{C}$

Hamming distance

The **Hamming distance** between two vectors is the Hamming weight of the difference of the vectors

$$d(\mathbf{v}, \mathbf{w}) = \omega(\mathbf{v} - \mathbf{w})$$

Minimum distance

The **minimum distance** of a code is the minimum distance among any couple of codewords

$$d_{min} = \min_{\mathbf{v}, \mathbf{w} \in \mathcal{C}, \mathbf{v} \neq \mathbf{w}} \omega(\mathbf{v} - \mathbf{w})$$

Decoding a code

Transmission errors

When travelling on a channel, the original codeword $\mathbf{x}$ might be corrupted, so that the received codeword is

$$\mathbf{y} = \mathbf{x} + \mathbf{e}, \text{ with } \mathbf{e} \text{ error vector}$$

For $\omega(\mathbf{e}) \leq \delta = \lfloor \frac{d_{min}-1}{2} \rfloor$ we are able to recover the original codeword

Decoding a code

Let $\mathbf{y} = \mathbf{x} + \mathbf{e}$ be the received codeword

Minimum distance decoding

Pick a codeword $\mathbf{u}$ that minimizes the Hamming distance $d(\mathbf{u}, \mathbf{y})$

$$\text{Decode}(\mathbf{y}) = \arg\min_{\mathbf{u} \in \mathcal{C}} d(\mathbf{u}, \mathbf{y})$$

Syndrome decoding

By the definition of syndrome of a codeword we have:

$$\mathbf{Hy} = \mathbf{H}(\mathbf{x} + \mathbf{e}) = \mathbf{Hx} + \mathbf{He} = \mathbf{0} + \mathbf{He} = \mathbf{He}$$

Hard problems of coding theory

Syndrome decoding problem [?]

Given a $[n, k]$ code $\mathcal{C}$ with parity matrix $\mathbf{H}$, the syndrome $\mathbf{y} \in \mathbb{F}_2^{n-k}$ and $w \leq n$, find the codeword $\mathbf{x} \in \mathbb{F}_2^n$ such that $\mathbf{H}\mathbf{x}^\top = \mathbf{y}^\top$ and $\omega(\mathbf{x}) = w$.

Hamming Quasi-Cyclic

Overview

HQC

One of the third round alternative candidates

- IND-CPA Public Key Encryption
- IND-CCA2 Key Encapsulation Mechanism
- Based on a variant of the syndrome decoding problem
- Employs quasi-cyclic codes to shorten key sizes

Codes

HQC employs two codes:

- a decodable$\mathcal{C}[n, k]$ generated by $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ (public)
- a random $[2n, n]$ code with parity-check matrix $\mathbf{H} = (\mathbf{I}_n | \mathbf{rot}(\mathbf{h}))$

Public Key Encryption

Setup

Generate the parameters $(n, k, \delta, w, w_r, w_e)$ for the corresponding security level λ

Key Generation

- Generate $seed_h \xleftarrow{\$} \{0,1\}^{\lambda}$
- Generate $\mathbf{h} \xleftarrow{seed_h} \mathbb{F}_2^n$
- Generate $(\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathbb{F}_2^n$, with $\omega(\mathbf{x}) = \omega(\mathbf{y}) = w$
- Calculate $\mathbf{s} = \mathbf{x} + \mathbf{hy}$

$\mathrm{pk} = (\mathbf{h}, \mathbf{s})$ $\mathrm{sk} = (\mathbf{x}, \mathbf{y})$

Public Key Encryption

To encrpyt a message $\mathbf{m} \in \mathbb{F}_2^k$:

Encryption

- Generate $(\mathbf{r_1}, \mathbf{r_2}) \xleftarrow{\$} \mathbb{F}_2^n$, with $\omega(\mathbf{r_1}) = \omega(\mathbf{r_2}) = w_r$
- Generate $\mathbf{e} \xleftarrow{\$} \mathbb{F}_2^n$, with $\omega(\mathbf{e}) = w_e$
- Calculate $\mathbf{u} = \mathbf{r_1} + \mathbf{h}\mathbf{r_2}$
- Calculate $\mathbf{v} = \mathbf{mG} + \mathbf{s}\mathbf{r_2} + \mathbf{e}$

$$\text{ctx} = (\mathbf{u}, \mathbf{v})$$

Public Key Encrpytion

Decryption

From the ctx $= (\mathbf{u}, \mathbf{v})$ and the private key pk $= (\mathbf{x}, \mathbf{y})$:

$$\begin{aligned}\mathbf{v} - \mathbf{u} \cdot \mathbf{y} &= (\mathbf{mG} + \mathbf{s} \cdot \mathbf{r_2} + \mathbf{e}) - (\mathbf{r_1} + \mathbf{h} \cdot \mathbf{r_2})\mathbf{y} = \mathbf{mG} + \mathbf{x} \cdot \mathbf{r_2} - \mathbf{r_1} \cdot \mathbf{y} + \mathbf{e} \\ &= \mathbf{mG} + (\mathbf{x} + \mathbf{h} \cdot \mathbf{y})\mathbf{r_2} + \mathbf{e} - (\mathbf{r_1} \cdot \mathbf{y} + \mathbf{h} \cdot \mathbf{r_2} \cdot \mathbf{y}) \\ &= \mathbf{mG} + \mathbf{x} \cdot \mathbf{r_2} + \mathbf{h} \cdot \mathbf{y} \cdot \mathbf{r_2} + \mathbf{e} - \mathbf{r_1} \cdot \mathbf{y} - \mathbf{h} \cdot \mathbf{r_2} \cdot \mathbf{y} \\ &= \mathbf{mG} + \mathbf{x_2} - \mathbf{r_1} \cdot \mathbf{y} + \mathbf{e}\end{aligned}$$

So we have $\mathbf{m} = \mathcal{C}.\mathrm{Decode}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$ whenever

$$\omega(\mathbf{x} \cdot \mathbf{r_2} - \mathbf{r_1} \cdot \mathbf{y} + \mathbf{e}) \leq \delta$$

Key Encapsulation Mechanism

By applying the Fujisaki-Okamoto transformation it is possible to build a IND-CCA2 KEM. Let $\mathcal{G}, \mathcal{H}, \mathcal{K}$ be hash functions and $\mathcal{E}$ an instance of HQC.PKE

- **Setup**: generate the parameters λ as in HQC.PKE, except that k will be the length of the key to be exchanged
- **KeyGen**: as in HQC.PKE

Key Encapsulation Mechanism

Let $\mathcal{G}, \mathcal{H}, \mathcal{K}$ be hash functions and $\mathcal{E}$ an instance of HQC.PKE

Encapsulation

- Generate the seed $\mathbf{m} \overset{\$}{\leftarrow} \mathbb{F}_2^k$ and the randomness $\theta \leftarrow \mathcal{G}(\mathbf{m})$
- Generate the ciphertext $\mathbf{c} \leftarrow (\mathbf{u}, \mathbf{v}) = \mathcal{E}.\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$
- Derive the symmetric key $K = \mathcal{K}(\mathbf{m}, \mathbf{c})$
- Calculate $\mathbf{d} \leftarrow \mathcal{H}(\mathbf{m})$
- Send the couple $(\mathbf{c}, \mathbf{d})$

Key Encapsulation Mechanism

Let $\mathcal{G}, \mathcal{H}, \mathcal{K}$ be hash functions and $\mathcal{E}$ an instance of HQC.PKE

Decapsulation

- Decrypt $\mathbf{m}^{'} \leftarrow \mathcal{E}.\text{Decrypt}(\text{sk}, \mathbf{c})$
- Compute $\theta^{'} \leftarrow \mathcal{G}(\mathbf{m}^{'})$ and $\mathbf{c}^{'} \leftarrow \mathcal{E}.\text{Encrpyt}(\text{pk}, \mathbf{m}^{'}, \theta^{'})$
- Check that $\mathbf{c} = \mathbf{c}^{'}$ and $\mathbf{d} = \mathcal{H}(\mathbf{m}^{'})$
- Derive the shared key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$

Masking

Side Channel Attacks

Implementations of theoretically secure cryptosystems may *leak* some information, allowing an attacker to extract secret data.

Side Channel Vulnerabilities

Side channel attacks take use of different leakage sources, such as:

- Execution time
- Power consumption
- Electromagnetic radiation
- Differential faults

Power Analysis

Simple Power Analysis

Analyze the electrical current drawn by the device over time: different operations may have different power consumption.

Differential Power Analysis

Statistically analyze the power consumption of the device over different executions: detect biases between, for example, a known secret key and a unknown one.

Power Analysis

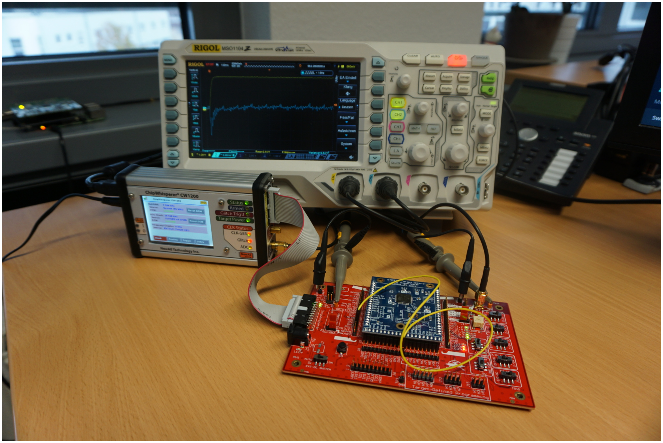


Figure: ChipWhisperer

Countermeasures to Power Analysis

Power analysis attacks are usually passive, so they cannot be detected by the device.

To mitigate the effectiveness of these attacks, it is possible to:

- **SPA**: avoid branches with conditions that depend on secret data
- **DPA**: *mask* the secret data when performing operations on them

Masking

Given a secret x, we can split it into d *shares* in such a way that

$$x = x_0 \odot x_1 \cdots \odot x_{d-1}$$

for some operation $\odot$ (e.g. XOR in binary fields).

Securing AND gates

Probing Model [ISW03]

Given an algorithm that operates on data split among d shares, we say that the algorithm is secure against a $(d-1)$-th order probing attack if, on input $x = (x_1, \cdots, x_d)$, it admits no tuple of $d-1$ (or less) shares that depends on x

Ishai-Sahai-Wagner's Scheme

Let $x = (x_1, \cdots, x_d), y = (y_1, \cdots, y_d)$ binary variables: to securely compute $x \wedge y$ at order $\lfloor d/2 \rfloor$:

- Pick a random bit $r_{i,j}$
- Compute $r_{j,i} = (r_{i,j} + (x_i \wedge y_j)) + (x_j \wedge y_i)$
- Compute $c_i = (x_i \wedge y_i) + \sum_{j \neq i} r_{i,j}$

Securing multiplications

The ISW scheme can be extended from $\mathbb{F}_2$ to $\mathbb{F}_2^n$, increasing its security to d-th order attacks

Rivain-Prouff's Scheme

Let $a = (a_1, \cdots, a_d), b = (b_1, \cdots, b_d)$, with $a_i, b_i \in \mathbb{F}_2^n$

- Calculate $c_i \leftarrow a_i \cdot b_i$ for each share

For i from 1 to d and j from $i+1$ to d:

- Extract a random value $s \xleftarrow{\$} \mathbb{F}_2^n$
- Calculate $s^{'} \leftarrow (s + (a_i \cdot b_j)) + (a_j \cdot b_i)$
- Calculate $c_i \leftarrow c_i + s$ and $c_j \leftarrow c_j + s^{'}$

The sum of the shares $(c_1, \cdots, c_d)$ yields the product $a \cdot b$

Implementation

Starting point

HQC has two software implementations

HQC implementations

- `reference`: implemented in C, no optimizations, not constant-time, sparse-dense multiplications
- `optimized`: implemented using AVX2 instructions, vectorized decoding, Karatsuba dense-dense multiplications

Our implementation starts from the `reference`.

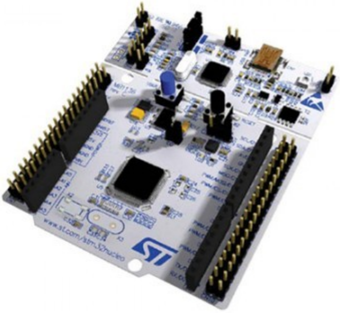
Target architecture

Our implementation is oriented torwards embedded devices

Target

Our target is the STM32F401RE board:

- Cortex-M4 processor (`ARMv7E` architecture)
- 512KB Flash Memory
- 96KB RAM
- No builtin RNG



Objects representations

- Random values: generated from a 40B seed with `seedexpander` functions
- Codewords: as polynomials (through arrays), in two possible forms:
  - dense: each bit represents a coefficient of the corresponding degree, implemented with `uint64_t` arrays of `VEC_N_SIZE_64` elements
  - sparse: elements correspond to the degrees with nonzero coefficient, implemented with `uint32_t` arrays of `PARAM_OMEGA` elements

Masking

Shares representations

The number of shares is a parameter of the cryptosystem, defined at compile time.
Shares are defined via the `shares_t` struct, containing a number of dense vectors equal to the number of shares

Splitting secrets into shares

The arrays containing secret data are divided in a number of blocks equal to the number of shares.
Each block is copied in the corresponding share, and elements outside the block are zeroed.

Operations on shares

Reducing shares to an array

When operations on secret data are completed, the shares of the secret element are added together with a XOR.

Adding shares

When adding up two `shares_t` a random mask is generated and *distributed* among the shares, so that by reducing to a single element the random mask cancels out.

Multiplication

In the optimized version the authors employ a Karatsuba dense-dense multiplication strategy

- Makes the operation time-constant (with some tweaks)
- Is significantly slower than the sparse-dense strategy

The sparsity of the operands (17669 vs 75) makes the preexisting sparse-dense multiplication more suitable in our case.

Sparse-dense multiplication

Variation of the schoolbook shift-and-add strategy.

Let A be the dense polynomial, B the sparse one, we have

$A \cdot X^{B_i} = A \cdot X^{B_i \bmod ts} \cdot X^{\lfloor \frac{B_i}{ts} \rfloor}$

Multiplication masking

`safe_mul`

Application of the Rivain-Prouff scheme:

- Operand size is reduced
- Apply polynomial reduction modulo $x^n + 1$ to multiplication results
- Loop unrolling

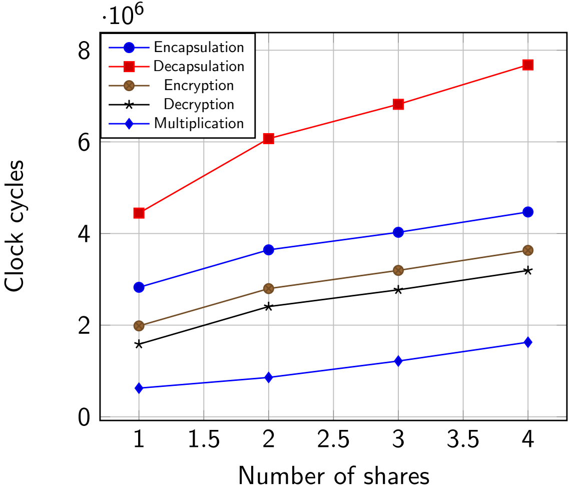
Where to apply `safe_mul`

The masked multiplication is more expensive, so it is applied only to calculate:

- Encryption: $\mathbf{s} \cdot \mathbf{r}_2$ (to calculate $\mathbf{v}$)
- Decryption: $\mathbf{u} \cdot \mathbf{y}$ (to calculate the codeword to decode)

Results

Execution time



Execution Time

Execution time (milliseconds)

#	encaps	decaps	enc	dec	mul
1	201.86	317.54	141.71	113.18	44.64
2	260.25	433.60	199.82	171.82	61.35
3	287.61	487.10	228.22	197.93	86.93
4	319.30	548.58	259.45	228.08	116.25

Percentual performance decrease

#	encaps	decaps	enc	dec	mul
2	28.92	36.55	41.00	51.81	37.42
3	42.48	53.40	61.05	74.88	94.73
4	58.18	72.76	83.09	101.51	160.41

Constant-time execution

Welch's t-test

Given two statistical populations X_1 and X_2 of N_1 and N_2 samples respectively:

$$t = \frac{\bar{X_1} - \bar{X_2}}{\sqrt{\frac{s^2_{X_1}}{N_1} + \frac{s^2_{X_2}}{N_2}}}$$

Can be used to verify that the means of the distributions are equal (null hypothesis).
Setting the confidence to 99.999%, we accept H_0 for $|t| \leq 4.5$

Constant-time execution

Test Vector Leakage Assessment

Based on the Welch's t-test, defines the two groups as:

- Random: use different inputs for each computation
- Fixed: use the same inputs for all computations

In case of random data (*e.g.* masks), these are always generated randomly.

Constant-time execution

Welch's t

#	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[a]	3.30161	4.01299	0.21834

[a] Not enough memory

References I

- Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg, *On the inherent intractability of certain coding problems (corresp.)*, IEEE Transactions on Information Theory **24** (1978), no. 3, 384–386.
- Yuval Ishai, Amit Sahai, and David Wagner, *Private circuits: Securing hardware against probing attacks*, Annual International Cryptology Conference, Springer, 2003, pp. 463–481.