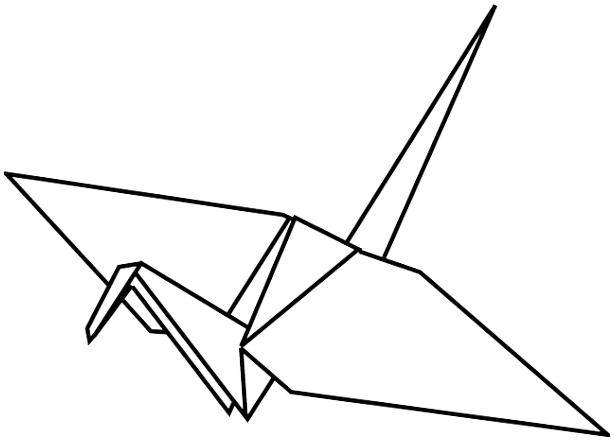
LEDAcrypt: Low-dEnsity parity-check coDe-bAsed cryptographic systems

Specification revision 2.0 – March 30, 2019



Name of the proposed cryptosystem

LEDAcrypt (Low-dEnsity parity-check coDe-bAsed cryptographic systems)

Submitters

This submission is from the following team, listed in alphabetical order:

- Marco Baldi, Università Politecnica delle Marche, Ancona, Italy
- Alessandro Barenghi, Politecnico di Milano, Milano, Italy
- Franco Chiaraluce, Università Politecnica delle Marche, Ancona, Italy
- Gerardo Pelosi, Politecnico di Milano, Milano, Italy
- Paolo Santini, Università Politecnica delle Marche, Ancona, Italy

E-mail addresses: `m.baldi@univpm.it`, `alessandro.barenghi@polimi.it`, `f.chiaraluce@univpm.it`, `gerardo.pelosi@polimi.it`, `p.santini@pm.univpm.it`.

Contact telephone and address

Marco Baldi (phone: +39 071 220 4894), Università Politecnica delle Marche, Dipartimento di Ingegneria dell'Informazione (DII), Via Brecce Bianche 12, I-60131, Ancona, Italy.

Names of auxiliary submitters

There are no auxiliary submitters. The principal submitter is the team listed above.

Name of the inventors/developers of the cryptosystem

Same as submitters.

Name of the owner, if any, of the cryptosystem

Same as submitters.

Backup contact telephone and address

Gerardo Pelosi (phone: +39 02 2399 3476), Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Via G. Ponzio 34/5, I-20133, Milano, Italy.

Signature of the submitter[1]

×

[1] See also printed version of "Statement by Each Submitter".

Contents

Acronyms

BF	Bit Flipping
BQP	Bounded-error Quantum Polynomial
CCA2	Adaptive Chosen Ciphertext Attack
DFR	Decoding Failure Rate
DRBG	Deterministic Random Bit Generator
GRS	Generalized Reed-Solomon
IND-CCA2	Indistinguishability Under Adaptive Chosen Ciphertext Attack
IND-CPA	Indistinguishability Under Chosen Plaintext Attack
ISD	Information Set Decoding
KEM	Key Encapsulation Module
KEM+DEM	Key Encapsulation Module + Data Encapsulation Mechanism
KI	Kobara-Imai
LDPC	Low-Density Parity-Check
NP	Nondeterministic-Polynomial
OW-CPA	One Wayness against Chosen Plaintext Attack
PFS	Perfect Forward Secrecy
PKC	Public-Key Cryptosystem
PKE	Public-Key Encryption
PRNG	Pseudo Random Number Generator
QROM	Quantum Random Oracle Model
QC	Quasi-Cyclic
QC-LDPC	Quasi-Cyclic Low-Density Parity-Check
QC-MDPC	Quasi-Cyclic Moderate-Density Parity-Check

ROM	Random Oracle Model
SDP	Syndrome Decoding Problem
TRNG	True Random Number Generator

Foreword

This document provides a complete and detailed specification of the post-quantum cryptographic primitives named LEDAcrypt (Low-dEnsity parity-check coDe-bAsed cryptographic systems), submitted to the 2nd round of NIST post-quantum contest [1]. LEDAcrypt is the result of the merger between the LEDAkem and LEDApkc proposals submitted to the 1st round of the contest [33], from which a Key Encapsulation Module (KEM) named LEDAcrypt KEM and a Public-Key Cryptosystem (PKC) named LEDAcrypt PKC have been respectively derived. In particular, LEDAcrypt takes into account the suggestions that were made in the NIST Internal Report 8240 in the tweaks made to the original submission.

Following the merger, LEDAcrypt includes:

- A revised specification of LEDAcrypt KEM with (compact) ephemeral keys (including minor tweaks and implementation optimizations), with Indistinguishability Under Chosen Plaintext Attack (IND-CPA) security guarantees. Parameters are proposed for security level 1, 3, and 5, and with three distinct rates of the underlying Quasi-Cyclic Low-Density Parity-Check (QC-LDPC) codes.
- A new specification of LEDAcrypt KEM with long-term keys aimed to key encapsulation mechanism + data encapsulation mechanism (KEM+DEM) applications, providing IND-CCA2 security guarantees. Parameters are proposed for security level 1, 3, and 5, with a single rate (i.e., $\frac{1}{2}$) of the QC-LDPC code for all of them.
- A revised specification of LEDAcrypt PKC with long-term keys as an alternate solution of the KEM+DEM one. optimizing bandwidth usage in case of "short" messages, providing IND-CCA2 security guarantees. Parameters are proposed for security level 1, 3, and 5, with a single rate (i.e., $\frac{1}{2}$) of the QC-LDPC code for all of them.

The following is a list of the tweaks applied to the LEDAcrypt cryptosystems:

- We took into account the possibility of performing an enumerative attack on a part of the secret key alone (namely, either one of the matrices H and Q) when selecting the parameters. While no practical attack takes advantage from this, we added this conservative measure to the parameter design procedure to provide a further hedge to attacks exploiting the additional structure of LEDAcrypt's secret codes.
- We introduced a rejection sampling phase during the key generation process to ensure that the weight of the product of the two secret matrices is maximal. While the odds of having a significant reduction in the weight of the product are definitely low, we maintain that the

computational cost of the rejection sampling is negligible, and so is the reduction to the keyspace. Such a rejection sampling procedure also permits us to provide a simpler analysis of the Decoding Failure Rate (DFR) of our codes and makes LEDAcrypt more prone to constant time implementation.

- We developed a theoretical characterization of the DFR for LEDAcrypt instances with long terms keys, avoiding the need of resorting to Monte Carlo simulations. Such a technique permits us to propose parameters such that the resulting code will have DFR low enough to provide IND-CCA2 guarantees for long term key pairs.

- The DFR analysis also relies on a variant of the Q-decoder we have proposed since the original submission, which always runs in two iterations. Such a feature removes the side channel leakage coming from the number of iterations made by the decoder.

- We employed one of the constructions reported in "A modular analysis of the Fujisaki-Okamoto transformation" by D. Hofheinz, K. Hövelmanns and E. Kiltz, (TCC 2017) to build LEDAcrypt KEM providing IND-CCA2 in the random oracle model. The same construction was proven to have IND-CCA2 guarantees in a subsequent work by H. Jiang, Z. Zhang and Z. Ma, "Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model", which is going to appear in PQCrypto 2019 (eprint available at `https://eprint.iacr.org/2019/134`).

- We provide an algorithmic parameter design procedure for LEDAcrypt, derived from the analysis reported in our last official comment, and taking into account a logarithmic cost of memory access in analyzing Information Set Decoding (ISD) algorithms.

- We provide an optimized implementation of LEDAcrypt KEM and LEDAcrypt PKC exploiting GCC intrinsics for the Intel AVX2 ISA extensions, while providing pure C99 fallback implementations to retain portability.

Chapter 1

Complete written specification

LEDAcrypt includes a KEM built from the Niederreiter cryptosystem, named LEDAcrypt KEM, and a PKC built from the McEliece cryptosystem, named LEDAcrypt PKC, both based on linear error-correcting codes. In particular, the following sets of instances are proposed:

i. One set of instances of LEDAcrypt KEM with ephemeral keys and IND-CPA, with additional resilience to accidental key reuse,

ii. One set of instances of LEDAcrypt KEM with long-lasting keys and IND-CCA2,

iii. One set of instances of LEDAcrypt PKC with long-lasting keys and IND-CCA2.

LEDAcrypt exploits the advantages of relying on QC-LDPC codes providing high decoding speeds and compact key pairs [3, 5], with the following main features:

i. A new decoding algorithm called Q-decoder is designed: it provides faster decoding than the classic Bit Flipping (BF) decoder, saves a computationally demanding matrix inverse computation, and allows a reduction in the required private key storage.

ii. A theoretical model is provided to predict the DFR due to decoding failures.

iii. LEDAcrypt KEM implements a KEM with IND-CPA and employs ephemeral keys to foil statistical attacks such as the one reported in [10].

iv. LEDAcrypt PKC implements a PKC with IND-CCA2 by employing a fully fledged conversion of the type described in [24, 25] and parameterizations that guarantee sufficiently low DFR.

v. Instances of LEDAcrypt KEM with sufficiently low DFR and IND-CCA2 are also proposed.

The main known attacks against these systems are those applicable against QC-LDPC code-based cryptosystems [3], which have been studied for twelve years since the first proposal appeared in [2], plus statistical attacks recently introduced in [10, 18]. We carefully analyze their capabilities and address parameterization for the LEDAcrypt primitives to provide the required security guarantees taking into account the computational cost reduction following from the use of a quantum computer in the solution of the underlying computationally hard problems.

1.1 Preliminaries

In this section we provide a set of background notions and nomenclature concerning finite fields and circulant matrix algebra, binary error correcting codes, and code-based cryptosystems, which will be employed in LEDAcrypt.

1.1.1 Finite fields and circulant matrix algebra

A $v \times v$ circulant matrix A is a matrix having the following form

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{v-1} \\ a_{v-1} & a_0 & a_1 & \cdots & a_{v-2} \\ a_{v-2} & a_{v-1} & a_0 & \cdots & a_{v-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_0 \end{bmatrix}. \tag{1.1}$$

According to its definition, any circulant matrix has a constant row and column weight, i.e., it is *regular*, since all its rows and columns are cyclic shifts of the first row and column, respectively.

A Quasi-Cyclic (QC) matrix is a matrix having the following form

$$B = \begin{bmatrix} B_{0,0} & B_{0,1} & \cdots & B_{0,w-1} \\ B_{1,0} & B_{1,1} & \cdots & B_{1,w-1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{z-1,0} & B_{z-1,1} & \cdots & B_{z-1,w-1} \end{bmatrix}, \tag{1.2}$$

where w and z are two positive integers and each block $B_{i,j}$ is a circulant matrix.

The set of $v \times v$ binary circulant matrices forms an algebraic ring under the standard operations of modulo-2 matrix addition and multiplication. The zero element is the all-zero matrix, and the identity element is the $v \times v$ identity matrix. The algebra of the polynomial ring $\mathbb{F}_2[x]/\langle x^v + 1\rangle$ is isomorphic to the ring of $v \times v$ circulant matrices over $\mathbb{F}_2$ with the following map

$$A \leftrightarrow a(x) = \sum_{i=0}^{v-1} a_i x^i. \tag{1.3}$$

According to (1.3), any binary circulant matrix is associated to a polynomial in the variable x having coefficients over $\mathbb{F}_2$ that coincide with the entries in the first row of the matrix

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{v-1} x^{v-1}. \tag{1.4}$$

In addition, according to (1.3), the all-zero circulant matrix corresponds to the null polynomial and the identity matrix to the unitary polynomial.

The ring of polynomials $\mathbb{F}_2[x]/\langle x^v + 1\rangle$ includes elements that are zero divisors: such elements are mapped onto singular circulant matrices over $\mathbb{F}_2$. Avoiding such matrices is important in the computation of the LEDAcrypt primitives, as non-singular $v \times v$ circulant matrices are required. However, a proper selection of the size v of a circulant matrix allows to easily generate invertible instances of it as described in the following.

Polynomial inversion in a finite field In order to provide efficient execution for the LEDAcrypt primitives, it is crucial to be able to efficiently check invertibility of a binary circulant matrix, and to generate a non-singular circulant matrix efficiently. To this end, we exploit the isomorphism (1.3) between $p \times p$ binary circulant matrices and polynomials in $\mathbb{F}_2[x]/\langle x^p+1\rangle$, turning the problem into providing an efficient criterion for the invertibility of an element of $\mathbb{F}_2[x]/\langle x^p+1\rangle$ and describing an efficient way to generate such invertible polynomials. In the following, we recall some facts from finite field theory, and we derive a necessary and sufficient condition for the invertibility of an element of $\mathbb{F}_2[x]/\langle x^p+1\rangle$, provided p is chosen according to the described criterion. Let $\mathbb{F}_{q^m}$ be the finite field of cardinality q^m, with q a prime power and m a positive integer; given an element $\alpha \in \mathbb{F}_{q^m}$, the following propositions hold [44]:

(i) The minimal polynomial of α with respect to $\mathbb{F}_q$, i.e., the nonzero monic polynomial $f(x) \in \mathbb{F}_q[x]$ of the least degree such that $f(\alpha) = 0$, always exists, it is unique, and it is also irreducible over $\mathbb{F}_q$.

(ii) If a monic irreducible polynomial $g(x) \in \mathbb{F}_q[x]$ has $\alpha \in \mathbb{F}_{q^m}$ as a root, then it is the minimal polynomial of α with respect to $\mathbb{F}_q$.

Definition 1.1.1 Let n be a positive integer and q a prime power such that $\texttt{gcd}(n, q) = 1$, which means that n and q are coprime. A cyclotomic coset of q mod n containing the value $a \in \mathbb{Z}_n$ is defined as

$$C_a = \{aq^j \bmod n : j = 0, 1, \ldots\}.$$

A subset $\{a_1, \ldots, a_s\} \subseteq \mathbb{Z}_n$ is named as a *complete set of representatives* of cyclotomic cosets of q mod n if $\forall\, i \neq j\ C_{a_i} \cap C_{a_j} = \varnothing$ and $\bigcup_j^s C_{a_j} = \mathbb{Z}_n$.

It is worth noting that the previous definition allows to easily infer that two cyclotomic cosets are either equal or disjoint. Indeed, given two cyclotomic cosets C_{a_1} and C_{a_2}, with $a_1 \neq a_2 \bmod n$, if $C_{a_1} \cap C_{a_2} \neq \varnothing$, two positive integers j and k such that $a_1 q^j = a_2 q^k \bmod n$ should exist. Assuming (without loss of generality) that $k \geq j$, the condition $\texttt{gcd}(n, q) = 1$ would ensure the existence of the multiplicative inverse of q and consequentially that $a_1 = a_2 q^{k-j} \bmod n$, which in turn would imply that the cyclotomic coset including a_1 is a subset of the coset including a_2, i.e., $C_{a_1} \subseteq C_{a_2}$. However, as the previous equality can be rewritten as $a_2 = a_1 (q^{-1})^{k-j} \bmod n$, it would also imply $C_{a_2} \subseteq C_{a_1}$, leading to conclude that $a_1 = a_2 \bmod n$, which is a contradiction of the initial assumption about them being different.

Two notable theorems that make use of the cyclotomic coset definition to determine the minimal polynomials of every element in a finite field can be stated as follows [44].

Theorem 1.1.1 Let α be a primitive element of $\mathbb{F}_{q^m}$, the minimal polynomial of α^i in $\mathbb{F}_q[x]$ is $g^{(i)}(x) = \prod_{j \in C_i} (x - \alpha^j)$, where C_i is the unique cyclotomic coset of q mod $q^m - 1$ containing i.

Theorem 1.1.2 Given a positive integer n and a prime power q, with $\texttt{gcd}(n, q) = 1$, let m be a positive integer such that $n \,|\, (q^m - 1)$. Let α be a primitive element of $\mathbb{F}_{q^m}$ and let $g^{(i)}(x) \in \mathbb{F}_q[x]$ be the minimal polynomial of $\alpha^i \in \mathbb{F}_{q^m}$. Denoting as $\{a_1, \ldots, a_s\} \subseteq \mathbb{Z}_n$ a complete set of

representatives of cyclotomic cosets of q mod n, the polynomial $x^n - 1 \in \mathbb{F}_q[x]$ can be factorized as the product of monic irreducible polynomials over $\mathbb{F}_q$, i.e.,

$$x^n - 1 = \prod_{i=1}^{s} g^{\left(\frac{(q^m-1)a_i}{n}\right)}(x).$$

Corollary 1.1.1 Given a positive integer n and a prime power q, with $\texttt{gcd}(n, q) = 1$, the number of monic irreducible factors of $x^n - 1 \in \mathbb{F}_q[x]$ is equal to the number of cyclotomic cosets of q mod n.

From the previous propositions on the properties of finite fields, it is possible to derive the following results.

Corollary 1.1.2 Given an odd prime number p, if 2 is a primitive element in the finite field $\mathbb{Z}_p$ then the irreducible (non trivial) polynomials being a factor of $x^p + 1 \in \mathbb{F}_2[x]$ are $x + 1$ and $\Phi(x) = x^{p-1} + x^{p-2} + \cdots + x + 1$.

Proof. Considering the ring of polynomials with binary coefficients $\mathbb{F}_2[x]$ and picking a positive integer n as an odd prime number (i.e., $n = p$), Corollary 1.1.1 ensures that the number of factors of $x^p + 1 \in \mathbb{F}_2[x]$ equals the number of cyclotomic cosets of 2 mod p.
If 2 is a primitive element of $\mathbb{Z}_p$, its order, $\text{ord}_p(2)$, is equal to the order of the (cyclic) multiplicative group of the field, i.e., $\text{ord}_p(2) = |\,(\mathbb{Z}_p \setminus \{0\}, \cdot)\,| = p - 1$; thus, the said cyclotomic cosets can be listed as: $C_0 = \{0 \cdot 2^j \bmod p : j = 0, 1, \dots\} = \{0\}$ and $C_1 = \{1 \cdot 2^j \bmod p : j = 0, 1, \dots\} = \mathbb{Z}_p \setminus \{0\}$.
The polynomial $x^p - 1 \in \mathbb{F}_2[x]$ admits $\alpha = 1$ as a root, therefore its two (non trivial) factors can be listed as: $x + 1$ and $\frac{x^p+1}{x+1} = x^{p-1} + x^{p-2} + \cdots + x + 1$. ■

Theorem 1.1.3 (Invertible elements in $\mathbb{F}_2[x]/\langle x^p + 1\rangle$) Let p be a prime number such that $\text{ord}_p(2) = p - 1$. Let $g(x)$ be a binary polynomial in $\mathbb{F}_2[x]/\langle x^p + 1\rangle$, with $\texttt{deg}(g(x)) > 0$.
$g(x)$ has a multiplicative inverse in $\mathbb{F}_2[x]/\langle x^p + 1\rangle$ if and only if it contains an odd number of terms and $g(x) \neq \Phi(x)$, with $\Phi(x) = x^{p-1} + x^{p-2} + \cdots + x + 1$.

Proof. If $g(x) \in \mathbb{F}_2[x]/\langle x^p + 1\rangle$ contains an odd number of terms and $g(x) \neq \Phi(x)$, to prove it is invertible modulo $x^p + 1$ we need to consider that $\texttt{gcd}(g(x), x^p + 1) = \texttt{gcd}(g(x), (x + 1)\Phi(x))$.
It is easy to observe that $x + 1$ does not divide $g(x)$, i.e., $(x + 1) \nmid g(x)$, as $g(1) = 1$, thus they are coprime. Considering $\Phi(x)$, we know by hypothesis that $\text{ord}_p(2) = p - 1$, therefore $\Phi(x)$ is irreducible over $\mathbb{F}_2[x]$ (see Corollary 1.1.2), which excludes that $g(x) \mid \Phi(x)$.
To the end of proving that $g(x)$ and $\Phi(x)$ are coprime, it has to hold that $\Phi(x) \nmid g(x)$. To this end assume, by contradiction, that $g(x)h(x) = \Phi(x)$ for a proper choice of $h(x) \in \mathbb{F}_2[x]$. The previous equality entails that $\texttt{deg}(g(x)) + \texttt{deg}(h(x)) = p - 1$, while $\texttt{deg}(g(x)) \leq p - 1$, which in turn leaves $\texttt{deg}(h(x)) = 0$ as the only option, leading to conclude $h(x) = 0$ or $h(x) = 1$. In case $h(x) = 0$, the equality $g(x) \cdot 0 = x^{p-1} + x^{p-2} + \cdots + x + 1$ is false, while in case $h(x) = 1$, the equality $g(x) \cdot 1 = \Phi(x)$ contradicts the hypothesis. Since we proved that $g(x) \nmid \Phi(x)$ and $\Phi(x) \nmid g(x)$, $g(x) \neq \Phi(x)$ by hypothesis, we can infer that they are coprime.
Finally, being $\texttt{gcd}(g(x), x^p + 1) = \texttt{gcd}(g(x), (x + 1)\Phi(x)) = 1$ we conclude that $g(x)$ is invertible.
To prove the other implication of the theorem, if $g(x) \in \mathbb{F}_2[x]/\langle x^p + 1\rangle$, with $\texttt{deg}(g(x)) > 0$, is invertible we need to derive that $g(x)$ must have an odd number of terms and be different from $\Phi(x)$. Being $g(x)$ invertible, this means that $\texttt{gcd}(g(x), x^p + 1) = \texttt{gcd}(g(x), (x + 1)\Phi(x)) = 1$, which in turn means that $\texttt{gcd}(g(x), x + 1) = 1$ and $\texttt{gcd}(g(x), \Phi(x)) = 1$ that guarantees that $g(x) \neq \Phi(x)$

and that $g(1) = 1$. Willing to prove that $g(x)$ must have an odd number of terms, assume, by contradiction, it has an even number of terms. Regardless of which terms are contained in $g(x)$ this means that it admits 1 as a root, which contradicts the premise. ■

Invertibility of square quasi cyclic binary matrices In the construction of the LEDAcrypt primitives we will need to generate QC binary square matrices that must be invertible. To remove the need to compute the matrix determinant, as it is computationally demanding, we prove and employ the following theorem. We denote as $\mathrm{perm}(\cdot)$ the permanent of a matrix, and as $w(\cdot)$ the number of the nonzero coefficients of a polynomial, a quantity also known as its weight.

Theorem 1.1.4 Let $p > 2$ be a prime such that $\mathrm{ord}_p(2) = p - 1$ and Q is an $n_0 \times n_0$ matrix of elements of $\mathbb{F}_2[x]/\langle x^p + 1\rangle$; if $\mathrm{perm}(w(Q))$ is odd and $\mathrm{perm}(w(Q)) < p$, then Q is non-singular.

Proof. Since each block Q_{ij} is isomorphic to a polynomial $q_{ij}(x) \in \mathbb{F}_2[x]/\langle x^p + 1\rangle$, the determinant of the matrix Q is represented as an element of $\mathbb{F}_2[x]/\langle x^p + 1\rangle$ as well. Let us denote by $d(x)$ the polynomial associated to the determinant. If the inverse of $d(x)$ exists, then Q is non-singular. According to Theorem 1.1.3, showing that $d(x)$ has odd weight and $d(x) \neq \Phi(x) = x^{p-1} + x^{p-2} + \cdots + 1$ is enough to guarantee that it is invertible. In general, when we are considering two polynomials $a(x)$ and $b(x)$, with $w(a(x)) = w_a$ and $w(b(x)) = w_b$, the following statements hold:

i. $w(a(x)b(x)) = w_a w_b - 2c_1$, where c_1 is the number of cancellations of pairs of monomials with the same exponent resulting from multiplication;

ii. $w(a(x) + b(x)) = w_a + w_b - 2c_2$, where c_2 is the number of cancellations due to monomials with the same exponent appearing in both polynomials.

The determinant $d(x)$ is obtained through multiplications and sums of the elements $q_{ij}(x)$ and, in case of no cancellations, has weight equal to $\mathrm{perm}(w(Q))$. If some cancellations occur, considering statements i) and ii) above, we have that $w(d(x)) = \mathrm{perm}(w(Q)) - 2c$, where c is the overall number of cancellations. So, even when cancellations occur, $d(x)$ has odd weight only if $\mathrm{perm}(w(Q))$ is odd. In addition, the condition $\mathrm{perm}(w(Q)) < p$ guarantees that $d(x) \neq \Phi(x)$, since $w(\Phi(x)) = p$. ■

With this result, we can guarantee that a QC matrix is non-singular, provided that the weights of its circulant blocks are chosen properly.

1.1.2 Quasi-cyclic low-density parity-check codes and their efficient decoding

Binary error correcting codes rely on a redundant representation of information in the form of binary strings to be able to detect and correct accidental bit errors which may happen during transmission or storage. We employ binary codes acting on a finite binary sequence at once, known as the information word, which are known as block codes. We will refer to them simply as binary codes in the following.

In this setting, let $\mathbb{F}_2$ be the binary finite field with the addition and multiplication operations that correspond to the usual exclusive-or and logical product between two Boolean values. Let $\mathbb{F}_2^k$ denote the k-dimensional vector space defined on $\mathbb{F}_2$. A binary code, denoted as $\mathcal{C}(n, k)$, is defined

as a bijective map $\mathcal{C}(n,k) : \mathbb{F}_2^k \to \mathbb{F}_2^n$, $n,k \in \mathbb{N}$, $0 < k < n$, between any binary k-tuple (i.e., an information word) and a binary n-tuple (denoted as codeword). The value n is known as the length of the code, while k is denoted as its dimension.
Encoding through $\mathcal{C}(n,k)$ means converting an information word $u \in \mathbb{F}_2^k$ into its corresponding codeword $c \in \mathbb{F}_2^n$. Given a codeword $\hat{c}$ corrupted by an error vector $e \in \mathbb{F}_2^n$ with Hamming weight $t > 0$ ($\hat{c} = c + e$), decoding instead recovers both the value of the information word u and the value of the error vector e. A code is said to be t-error correcting if, for any value of e, given $\hat{c}$ there is a decoding procedure to retrieve both the error vector e and the original information word u.

Definition 1.1.2 (Linear Code) The code $\mathcal{C}(n,k)$ is linear if and only if the set of its 2^k codewords is a k-dimensional subspace of the vector space $\mathbb{F}_2^n$.

A property of linear block codes that follows from Definition 1.1.2 is that the sum modulo 2, i.e., the component wise exclusive-or, of two codewords is also a codeword.

Definition 1.1.3 (Minimum distance) Given a linear binary code $\mathcal{C}(n,k)$, the minimum distance of $\mathcal{C}(n,k)$ is the minimum Hamming distance among all the ones which can be computed between a pair of its codewords.

If the code is linear, its minimum distance coincides with the minimum Hamming weight of its nonzero codewords.

Definition 1.1.4 (Encoding) Given $\mathcal{C}(n,k)$, a linear error correcting code, and $\Gamma \subset \mathbb{F}_2^n$ the vector subspace containing its 2^k codewords, it is possible to represent it choosing k linearly independent codewords $\{g_0, g_1, \ldots g_{k-1}\} \in \mathbb{F}_2^n$ to form a basis of Γ. Any codeword $c = [c_0, c_1, \ldots, c_{n-1}]$ can be expressed as a linear combination of the vectors of the basis

$$c = u_0 g_0 + \mathrm{u}_1 g_1 + \ldots + u_{k-1} g_{k-1}, \tag{1.5}$$

where the binary coefficients u_i can be thought as the elements of an information vector $u = [u_0, u_1, \ldots, u_{k-1}]$, which the code maps into c. We then say that u is encoded into c.

Equation (1.5) can be rewritten as $c = uG$, where G is a $k \times n$ binary matrix known as the generator matrix of the code $\mathcal{C}(n,k)$, i.e.,

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix}.$$

Since any set of k linearly independent codewords can be used to form G, a code can be represented by different generator matrices. Among the possible generator matrices for a linear code, one known as systematic can always be derived.

Definition 1.1.5 (Systematic Encoding) A linear error correcting code $\mathcal{C}(n,k)$ is said to have systematic encoding, or to be systematic in short, if each one of its codewords contains the information vector it is associated to.

A conventional way to express a systematic code is the one where each n-bit codeword, c, is obtained by appending $r = n - k$ redundancy bits $(c_k, c_{k+1}, \ldots, c_{n-1})$ to its corresponding k-bit information word (i.e., $c_0, c_1, \ldots, c_{k-1}$, with $c_i = u_i$, $0 \leq i < k$): $c = [u_0, u_1, \ldots, u_{k-1} | c_k, c_{k+1}, \ldots, c_{n-1}]$. It follows that the associated $k \times n$ generator matrix G can be written as $G = [I_k | P]$, where I_k denotes the $k \times k$ identity matrix and P is a $k \times r$ binary matrix.
Let us consider the set of all n-bit vectors in $\mathbb{F}_2^n$ that are orthogonal to any codeword of the code subspace Γ, known as its orthogonal complement $\Gamma^{\perp}$. Its dimension is $\dim\left(\Gamma^{\perp}\right) = n - \dim(\Gamma) = n - k = r$. A basis of $\Gamma^{\perp}$ is readily obtained choosing r linearly independent vector in $\Gamma^{\perp}$ as

$$H = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{r-1} \end{bmatrix}.$$

The $r \times n$ matrix H is known as a parity-check matrix of the code $\mathcal{C}(n, k)$, while, for any n-bit vector $x \in \mathbb{F}_2^n$, the $r \times 1$ vector $s = Hx^T$, where ^{T} denotes transposition, is known as the syndrome of x through H. Given that H is a basis of $\Gamma^{\perp}$, every codeword $c \in \Gamma$ satisfies the equality $Hc^T = 0_{r \times 1}$ where $0_{r \times 1}$ is the $r \times 1$ zero vector, i.e., a codeword belonging to $\mathcal{C}(n, k)$ has a null syndrome through H.

It is easy to show that the generator matrix G and the parity-check matrix H are two equivalent descriptions of a linear code. Indeed, we have that $Hc^T = HG^T u^T = 0_{r \times 1}$, $\forall\, u \in \mathbb{F}_2^k$, yielding in turn that $HG^T = 0_{r \times k}$. Exploiting the aforementioned relation, it is possible to derive H from G and vice versa. Let us consider, for the sake of clarity, the case of a systematic code $\mathcal{C}(n, k)$ with $G = [I_k | P]$. It is possible to obtain the corresponding parity-check matrix H as $\left[P^T | I_r\right]$, which satisfies $HG^T = P^T + P^T = 0_{r \times k}$. Finally, considering a generic parity-check matrix $H = [A|B]$, with A an $r \times k$ matrix and B an $r \times r$ non-singular matrix, a systematic generator matrix of the corresponding code is computed as $G = \left[I_k | \left(B^{-1}A\right)^T\right]$, being B^{-1} the inverse of matrix B.

A QC code is defined as a linear block code $\mathcal{C}(n, k)$ having information word size $k = pk_0$ and codeword size $n = pn_0$, where n_0 is denoted as *basic block length* of the code and each cyclic shift of a codeword by n_0 symbols results in another valid codeword [42].
LEDAcrypt hinges on a QC code $\mathcal{C}(pn_0, pk_0)$ having the generator and parity-check matrices composed by $p \times p$ circulant sub-matrices (blocks).

A Low-Density Parity-Check (LDPC) code $\mathcal{C}(n, k)$ is a special type of linear block code characterized by a sparse parity-check matrix H. In particular, the Hamming weight of a column of H, denoted as d_v, is much smaller than its length r and increases sub-linearly with it. In terms of error correction capability, LDPC codes having a non-constant weight for either the rows or the columns of H, hence known as irregular LDPC codes, were proven to approach the channel capacity [29]. Considering the parity-check matrix H of an LDPC code as the incidence matrix of a graph, such a graph is known as Tanner graph, and it has been shown that keeping the number of short cycles as small as possible in such a graph is beneficial to the error correction performance of the code.

The peculiar form of LDPC codes allows to devise an efficient decoding procedure, provided their parity-check matrix H is known, via algorithms known as BF decoders [13]. Indeed, BF algorithms perform decoding with a fixed-point procedure which exploits the form of H to iteratively deduce which bits of an error-affected codeword should be flipped in order to obtain a zero-valued syndrome for it. If the fixed-point procedure converges within a desired amount of iterations to a zero-valued syndrome, the decoding action is deemed successful.

The rationale of BF decoders is in considering the parity-check matrix H as the description of a set of r equations in the codeword bits yielding the syndrome bits as their results. Such equations are known as parity-check equations, or parity checks, in short. In this context, the one-valued coefficients of the i-th column of a parity matrix H can be thought of as the indicators of which parity checks of the code are involving the i-th bit of the received codeword. The result of each one of the said parity checks is a bit in the syndrome, hence a zero-valued syndrome indicates a set of successful parity checks, and thus a correct codeword. The convergence of the fixed-point decoder is influenced by the number of parity checks in which each codeword element is involved: in particular, being involved in a small number of parity checks speeds up the convergence.

An LDPC code may also be a QC code, expressed with a QC parity-check or generator matrix, hence being named a QC-LDPC code, which is indeed the case of the codes employed in LEDAcrypt.

An efficient BF decoding procedure for QC-LDPC codes can be devised relying on the number of unsatisfied parity checks to which a codeword bit concurs as an estimate of it being affected by an error. We describe such a procedure in Algorithm 1, where the sparse and QC nature of the matrix H is explicitly exploited.

To this end H is represented as $r_0 \times n_0$ sparse $p \times p$ circulant blocks, and only the positions of the first column of each block are memorized in `Hsparse`. Algorithm 1 receives, alongside `Hsparse`, the error-affected codeword to be corrected x, its syndrome computed as $s = Hx^T$, and performs the fixed-point decoding procedure for a maximum of `imax` iterations. The algorithm outputs its best estimate for the correct codeword c and a boolean variable `decodeOk` reporting the success of the decoding procedure. The procedure iterates at fixed-point (loop at lines 4–18) the decoding procedure, which starts by counting how many unsatisfied parity checks a codeword bit is involved into (lines 5–10). Such a value is obtained considering which are the asserted bits in a given column of H, taking care of accounting for its sparse representation, and the cyclic nature of its blocks (line 8). Whenever a bit in the i-th column and `assertedHbitPos`-th row of H is set, it is pointing to the fact that the i-th bit of the codeword is involved in the `assertedHbitPos`-th parity-check equation. Thus, if the `assertedHbitPos`-th bit of the syndrome is unsatisfied, i.e., equal to 1, the number of unsatisfied parity checks of the i-th bit is incremented (lines 9–10).

Once the computation of the number of unsatisfied parity checks per codeword bit is completed, a decision must be taken on which of them are to be flipped, as they are deemed error affected. The choice of the threshold that the number of unsatisfied parity checks should exceed, can be done a priori from the code parameters, or determined taking into account the iteration reached by the decoder and the current weight of the syndrome.

Thus, the procedure toggles the values of all the codeword bits for which the number of unsatisfied parity checks matches the maximum one (lines 12–14). Once this step is completed, the values of the parity checks should be recomputed according to the new value of the codeword. While this can be accomplished by pre-multiplying the transposed codeword by H, it is more efficient to exploit the knowledge of which bits of the codeword were toggled to change only the parity-check values in the syndrome affected by such toggles. Lines 15–17 of Algorithm 1 update the syndrome according to the aforementioned procedure, i.e., for a given i-th codeword bit being toggled, all the syndrome values corresponding to the positions of the asserted coefficients in the i-th column of H are also toggled. Once either the decoding procedure has reached its intended fixed-point, i.e., the syndrome is a zero-filled vector, or the maximum number of iterations has been reached, Algorithm 1 returns its best estimate for the corrected codeword, together with the outcome of the decoding procedure (lines 19–21).

Algorithm 1: BF decoding

Input: x: QC-LDPC error-affected codeword as a $1 \times pn_0$ binary vector.
s: QC-LDPC syndrome. It is a $pr_0 \times 1$ binary vector obtained as $s = Hx^T$.
`Hsparse`: sparse version of the parity-check matrix H, represented as a $d_v \times n_0$ integer matrix containing for each of its n_0 columns, the positions in $\{0, 1, \ldots, pr_0 - 1\}$ of the asserted binary coefficients in the first column of the sequence of r_0 circulant block matrices (any of which with size $p \times p$).

Output: c: error-free $1 \times pn_0$ codeword
`decodeOk`: Boolean value denoting the successful outcome of the decoding action

Data: `imax`: the maximum number of allowed iterations before reporting a decoding failure

```
1  codeword ← x                                              // bitvector with size pn_0
2  syndrome ← s                                              // bitvector with size pr_0
3  iter ← 0                  // scalar variable denoting the number of iterations
4  repeat
5      iter ← iter + 1
6      unsatParityChecks ← 0_{1×pr_0} // counters of unsatisfied parity checks
7      for i = 0 to n_0 − 1 do
8          for exp = 0 to p − 1 do
9              for j = 0 to d_v − 1 do
10                 assertedHbitPosition ←
                    (exp + Hsparse[j][i]) mod p + p · ⌊Hsparse[j][i] div p⌋
11                 if syndrome[assertedHbitPosition] = 1 then
12                     unsatParityChecks[ip + exp] ← 1 + unsatParityChecks[ip + exp]
13     threshold ← ThresholdChoice(iterationCounter, syndrome)
14     for i = 0 to pn_0 − 1 do
15         if unsatParityChecks[i] ≥ threshold then
16             BitToggle(codeword[i])                                  // codeword update
17             for j = 0 to d_v − 1 do
18                 assertedHbitPos ← (exp + Hsparse[j][i]) mod p + p · ⌊Hsparse[j][i] div p⌋
19                 BitToggle(syndrome[assertedHbitPos])
20 until syndrome ≠ 0_{1×pr_0}  AND  iter < imax
21 if syndrome = 0_{1×pr_0} then
22     return codeword, true
23 return codeword, false
```

1.1.3 Classic code-based cryptosystems

The McEliece cryptosystem is a public-key encryption (PKE) scheme proposed by Robert McEliece in 1978 [31] and exploiting the hardness of the problem of decoding a random-like linear block code. In the original proposal, the McEliece cryptosystem used irreducible Goppa codes as secret codes, but its construction can be generalized to other families of codes. The triple of polynomial-time algorithms $\Pi^{\texttt{McE}}$=($\texttt{Keygen}^{\texttt{McE}}$, $\texttt{Enc}^{\texttt{McE}}$, $\texttt{Dec}^{\texttt{McE}}$) defining the scheme are as follows:

- The *key-generation* algorithm considers a binary linear block code $\mathcal{C}(n,k)$, with codeword length n, information word length k and outputs a secret key $sk^{\texttt{McE}}$ defined as the generator matrix $G_{k\times n}$ of a code $\mathcal{C}(n,k)$ able to correct $t \geq 1$ or less bit errors, plus a randomly chosen invertible binary matrix $S_{k\times k}$, named scrambling matrix, and a binary permutation matrix $P_{n\times n}$:

$$sk^{\texttt{McE}} \leftarrow \{S, G, P\} \tag{1.6}$$

  The corresponding public key $pk^{\texttt{McE}}$ is computed as the generator matrix $G'_{k\times n}$ of a permutation-equivalent code with the same size and correction capability of the original code:

$$pk^{\texttt{McE}} \leftarrow \{G'\}, \text{ with } G' = SGP \tag{1.7}$$

- The *encryption* algorithm takes as input a public key $pk^{\texttt{McE}}$ and a message composed as a $1 \times k$ binary vector u, and outputs a ciphertext $x \leftarrow \texttt{Enc}^{\texttt{McE}}\left(pk^{\texttt{McE}}, u\right)$ computed as:

$$x = uG' + e, \tag{1.8}$$

  where e is a $1 \times n$ random binary error vector with weight t (i.e., with exactly t asserted bits).

- The *decryption* algorithm takes as input a secret key $sk^{\texttt{McE}}$ and a ciphertext $x_{1\times n}$ and outputs a message $m' \leftarrow \texttt{Dec}^{\texttt{McE}}\left(sk^{\texttt{McE}}, x\right)$ computed as the result of a known error correction decoding algorithm (`Decode`) able to remove t errors present in xP^{-1} and subsequently multiplying by the inverse of the matrix S:

$$x' = \texttt{Decode}(xP^{-1})S^{-1} = \texttt{Decode}\left((uS)G + (eP^{-1})\right)S^{-1} = (uS)S^{-1} = u \tag{1.9}$$

  In the original McEliece cryptosystem algebraic code families (namely, Goppa codes) provided with bounded-distance decoders were used. In such a case, since the number of errors correctable by the secret code is t, the correction of the error vector eP^{-1} is ensured by design and the cryptosystem exhibits a zero DFR.

  It is also worth noticing that the original McEliece cryptosystem only provides One Wayness against Chosen Plaintext Attack (OW-CPA) guarantees, which means that given a ciphertext it is computationally impossible to recover the plaintext without knowing the private key. Suitable conversions of the cryptosystem must be exploited in order to achieve IND-CCA2, which means that an adversary with access to a decryption oracle (that knows the private key) cannot distinguish whether a string is a decryption of a legitimate given ciphertext or a random valid plaintext message. The decryption oracle cannot be queried on the given ciphertext. When these conversions are used, some constraints on the public code can be relaxed.

The Niederreiter cryptosystem [34] is a code-based cryptosystem exploiting the same trapdoor introduced in the McEliece PKE [31] with an alternative formulation. The Niederreiter PKE employs syndromes and parity-check matrices in place of the codewords and generator matrices employed by the algorithms in the McEliece PKE. The first proposal of such a scheme employed Generalized Reed-Solomon (GRS) codes that were proven to make the whole construction vulnerable. However, when the same family of codes is used, Niederreiter and McEliece cryptosystems exhibit the same cryptographic guarantees [28]. The triplet of polynomial-time algorithms $\Pi^{\texttt{Nie}}$=($\texttt{Keygen}^{\texttt{Nie}}$, $\texttt{Enc}^{\texttt{Nie}}$, $\texttt{Dec}^{\texttt{Nie}}$) defining the scheme are as follows:

- The *key-generation* algorithm considers a binary linear block code $\mathcal{C}(n,k)$, with codeword length n, information word length k and outputs a secret key $sk^{\texttt{Nie}}$ defined as the parity-check matrix $H_{r\times n}$ of a code $\mathcal{C}(n,k)$, $r=n-k$ able to correct $t\geq 1$ or less bit errors, plus a randomly chosen invertible binary matrix $S_{r\times r}$, named scrambling matrix:

$$sk^{\texttt{Nie}} \leftarrow \{H, S\} \tag{1.10}$$

  The corresponding public key $pk^{\texttt{Nie}}$ is computed as the parity-check matrix $H'_{r\times n}$ obtained as the product of the two secret matrices and is equivalent to H:

$$pk^{\texttt{Nie}} \leftarrow \{H'\}, \text{ with } H' = SH \tag{1.11}$$

  Note that the knowledge of H' is not amenable to be employed with an efficient decoding algorithm as it actually hides the structure of the selected code.

- The *encryption* algorithm takes as input a public key $pk^{\texttt{Nie}}$ and a message composed as a $1\times n$ binary vector e with exactly t asserted bits, and outputs a ciphertext $x_{r\times 1} \leftarrow \texttt{Enc}^{\texttt{Nie}}\left(pk^{\texttt{Nie}}, e\right)$ computed as the syndrome of the original message:

$$x = H'e^T = SHe^T \tag{1.12}$$

- The *decryption* algorithm takes as input a secret key $sk^{\texttt{Nie}}$ and a ciphertext $x_{r\times 1}$ and outputs a message $e_{1\times n} \leftarrow \texttt{Dec}^{\texttt{Nie}}\left(sk^{\texttt{Nie}}, x\right)$ computed as the result of a known error correction syndrome decoding algorithm (`SynDecoding`) applied to the vector $S^{-1}x$ and able to recover the original error vector $e_{1\times n}$ (as well as the zero codeword $0_{1\times n}$):

$$e = \texttt{SynDecoding}(S^{-1}x) = \texttt{SynDecoding}(He^T) \tag{1.13}$$

  In the original Niederreiter cryptosystem algebraic code families provided with bounded-distance decoders were considered. In such a case, the syndrome decoding algorithm allows to deterministically recover the original error vector with weight t and the cryptosystem exhibits a zero DFR.

1.2 QC-LDPC code-based McEliece and Niederreiter cryptosystems

In the following we describe the algorithms of both McEliece $\Pi^{\texttt{McE}}$=($\texttt{Keygen}^{\texttt{McE}}$, $\texttt{Enc}^{\texttt{McE}}$, $\texttt{Dec}^{\texttt{McE}}$) and Niederreiter $\Pi^{\texttt{Nie}}$=($\texttt{Keygen}^{\texttt{Nie}}$, $\texttt{Enc}^{\texttt{Nie}}$, $\texttt{Dec}^{\texttt{Nie}}$) cryptosystems instantiated with QC-LDPC codes.

- The *key-generation* algorithms $\textsc{Keygen}^{\texttt{Nie}}$ in Fig. 1.1(a) and $\textsc{Keygen}^{\texttt{McE}}$ in Fig. 1.1(b) consider a QC-LDPC code $\mathcal{C}(n,k)$, with codeword length $n=pn_0$, information word length $k=p(n_0-1)$, where $n_0\in\{2,3,4\}$, p is a prime number such that $\texttt{ord}_p(2)=p-1$.

  The first step in the key generation procedures is to find two random binary matrices that correspond to the (secret) quasi-cyclic $p\times pn_0$ parity-check matrix H of a QC-LDPC code with the mentioned parameters and a $pn_0\times pn_0$ quasi-cyclic sparse binary matrix Q. The matrix H is structured as $1\times n_0$ circulant blocks, each of which with size $p\times p$ and with a fixed

Algorithm 2: KEYGEN$^{\texttt{Nie}}$

Output: $(sk^{\texttt{Nie}}, pk^{\texttt{Nie}})$
Data: p>2 prime, $\texttt{ord}_p(2)$=$p-1$, $n_0 \geq 2$

1 $\texttt{seed} \leftarrow$ TRNG()
2 $\{H, Q\} \leftarrow$ GENERATEHQ(seed)
// $H = [H_0, \ldots, H_{n_0-1}]$,
// H_i is a $p \times p$ circulant block
// with $w(H_i) = d_v, 0 \leq i < n_0$
// Q is a $n_0 \times n_0$ block matrix
// in accordance to Th. 1.1.4
// $w([Q_{i,0}, \ldots, Q_{i,n_0-1}])$=$m$,0≤$i$<$n_0$
3 $L \leftarrow HQ$
// $L = [L_0, L_1, \ldots, L_{n_0-1}]$,
// $L_j = \sum_i H_i Q_{ij}$ is a $p \times p$ circulant block
4 **if** $\exists\ 0 \leq j < n_0$ **s.t.** $w(L_j) \neq d_v \times m$ **then**
5 **goto** 2
6 LInv $\leftarrow$ COMPUTEINVERSE(L_{n_0-1})
7 **for** $i = 0$ **to** $n_0 - 2$ **do**
8 $M_i \leftarrow \texttt{LInv}\, L_i$
// M_i is a $p \times p$ circulant block
9 $pk^{\texttt{Nie}} \leftarrow [M_0 \mid \ldots \mid M_{n_0-2} | I]$
10 $sk^{\texttt{Nie}} \leftarrow \{H, Q\}$
// I is a $p \times p$ identity block
11 **return** $(sk^{\texttt{Nie}}, pk^{\texttt{Nie}})$

(a)

Algorithm 3: KEYGEN$^{\texttt{McE}}$

Output: $(sk^{\texttt{McE}}, pk^{\texttt{McE}})$
Data: p>2 prime, $\texttt{ord}_p(2)$=$p-1$, $n_0 \geq 2$

1 $\texttt{seed} \leftarrow$ TRNG()
2 $\{H, Q\} \leftarrow$ GENERATEHQ(seed)
// $H = [H_0, \ldots, H_{n_0-1}]$,
// H_i is a $p \times p$ circulant block
// with $w(H_i) = d_v, 0 \leq i < n_0$
// Q is a $n_0 \times n_0$ block matrix
// in accordance to Th. 1.1.4
// $w([Q_{i,0}, \ldots, Q_{i,n_0-1}])$=$m$,0≤$i$<$n_0$
3 $L \leftarrow HQ$
// $L = [L_0, L_1, \ldots, L_{n_0-1}]$,
// $L_j = \sum_i H_i Q_{ij}$ is a $p \times p$ circulant block
4 **if** $\exists\ 0 \leq j < n_0$ **s.t.** $w(L_j) \neq d_v \times m$ **then**
5 **goto** 2
6 LInv $\leftarrow$ COMPUTEINVERSE(L_{n_0-1})
7 **for** $i = 0$ **to** $n_0 - 2$ **do**
8 $M_i \leftarrow \texttt{LInv}\, L_i$
// M_i is a $p \times p$ circulant block
9 $Z \leftarrow \texttt{diag}([I, \ldots, I])$
// $p(n_0-1)\times p(n_0-1)$ identity
// matrix, composed as a diagonal
// block matrix with $(n_0 - 1)$
// replicas of I, where I is
// a $p \times p$ identity matrix
10 $pk^{\texttt{McE}} \leftarrow \left[Z \mid [M_0 \mid \ldots \mid M_{n_0-2}]^T\right]$
11 $sk^{\texttt{McE}} \leftarrow \{H, Q\}$
12 **return** $(sk^{\texttt{McE}}, pk^{\texttt{McE}})$

(b)

Figure 1.1: Summary of the key generation algorithms of both Niederreiter (a) and McEliece (b) cryptoschemes, instantiated with QC-LDPC codes

odd number of asserted elements per row/column, denoted as d_v (to guarantee invertibility of each block – see Th. 1.1.3):

$$H = [H_0, H_1, \ldots, H_{n_0-1}],\ w(H_i) = d_v,\ 0 \leq i < n_0.$$

The matrix Q is structured as a $n_0 \times n_0$ block matrix, where each block is a $p \times p$ binary circulant matrix (note that the existence of a multiplicative inverse of any of these blocks is not guaranteed). The weights of each block of Q define a circulant matrix of integers denoted

as $w(Q)$ such that the sum of all elements in any row/column of Q amounts to the same value $m = \sum_{i=0}^{n_0-1} m_i$.

$$Q = \begin{bmatrix} Q_{0,0} & Q_{0,1} & \cdots & Q_{0,n_0-1} \\ Q_{1,0} & Q_{1,1} & \cdots & Q_{1,n_0-1} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{n_0-1,0} & Q_{n_0-1,1} & \cdots & Q_{n_0-1,n_0-1} \end{bmatrix} \quad w(Q) = \begin{bmatrix} m_0 & m_1 & \cdots & m_{n_0-1} \\ m_{n_0-1} & m_0 & \cdots & m_{n_0-2} \\ \vdots & \vdots & \ddots & \vdots \\ m_1 & m_{n_0-1} & \cdots & m_0 \end{bmatrix}.$$

The choice of the weights $m_0, m_1, \cdots, m_{n_0-1}$ is constrained according to Theorem 1.1.4 to ensure the invertibility of Q, even if any individual block may not be invertible.

In computing the product HQ (see line 3 of both Fig. 1.1(a) and Fig. 1.1(b)), the multiplication of H by the (compatible) full-rank matrix Q ($\mathtt{rank}(Q) = pn_0$) yields a matrix $L = HQ$ with the same rank of H, thus $\mathtt{rank}(L) = p$. This implies that every $p \times p$ block of $L = [L_0, L_1, \ldots, L_{n_0-1}]$ has the same rank (i.e., $\mathtt{rank}(L_i) = p$) and is therefore invertible. As a consequence in line 6 of both Fig. 1.1(a) and Fig. 1.1(b), there is no need to check whether L_{n_0-1} admits a multiplicative inverse or not.

It is worth noting that the necessary and sufficient conditions in Th. 1.1.3 state that an invertible binary circulant block matrix must exhibit a weight of any row/column equal to an odd number. Since $L_j = \sum_{i=0}^{n_0-1} H_i Q_{ij}$, the weight of L_j satisfies also the following $w(L_j) = d_v \times m - 2c$, where the parameter $c \geq 0$ is justified with an argument analogous to the one reported in the proof of Th. 1.1.4. From this, it is easy to conclude that $m = \sum_{i=0}^{n_0-1} m_i$ must also be an odd number.

Lines 4–5 in both Fig. 1.1(a) and Fig. 1.1(b) check that all the blocks of the matrix L have a weight equal to $d_v \times m$, and repeat the generation process until this condition does hold. Such a constraint is imposed as the weight of the blocks of L is a security parameter in LEDAcrypt, and then must be appropriately constrained. Considerations on the probability of drawing instances of the matrices H and Q that match the said criterion are made in Section A.

Starting from the multiplicative inverse of L_{n_0-1}, the following matrix can be computed

$$M = L_{n_0-1}^{-1} L = [M_0|M_1|M_2|\ldots|M_{n_0-2}|I_p] = [M_l|I] \tag{1.14}$$

where I denotes the $p \times p$ identity matrix. The matrix M in (1.14) is the parity-check matrix of the public code in systematic form, and can easily be converted into the systematic generator matrix of the same code.

The private key for each of the schemes at hand consists of the two matrices H and Q: $sk^{\mathtt{Nie}} \leftarrow \{H, Q\}$, $sk^{\mathtt{McE}} \leftarrow \{H, Q\}$. The computation of the public key depends on whether the scheme is Niederreiter-based or McEliece-based. Specifically, the computation of the public key values will yield either a parity-check matrix or a generator matrix, both in systematic form, referring to a public code without a computationally efficient decoding/decryption algorithm (see lines 7–9 in both Fig. 1.1(a) and Fig. 1.1(b)).

It is worth noting that in practice there is no need to store a public key including circulant matrices (blocks) that equal to an identity $p \times p$ matrix. Moreover, since both H and Q are formed by sparse circulant matrices (blocks), it is convenient to store each block as the set of integers representing the positions of non-zero elements of the first row of each block. The said set of integers requires at least $\lceil \log_2(p) \rceil$ bits to be stored.

Algorithm 4: ENCRYPT$^{\texttt{Nie}}$

Input: $e = [e_0, \ldots, e_{n_0-1}]$: plaintext message; randomly chosen $1 \times pn_0$ binary vector, with t asserted bits, where each e_j is a $1\times p$ vector with $0\leq j<n_0$.
$pk^{\texttt{Nie}} = [M_0 \mid \ldots \mid M_{n_0-2}|I]$ public key: sequence of n_0-1 $p\times p$ circulant blocks M_j, with $0\leq j<n_0-1$, followed by an identity block

Output: s: syndrome; $p \times 1$ binary vector

Data: $p>2$ prime, $\texttt{ord}_p(2)=p-1$, $n_0 \geq 2$

$s \leftarrow 0_{p\times 1}$ `// zero vector`
for $j = 0$ **to** $n_0 - 1$ **do**
    $s \leftarrow s + M_j\, e_j^T$
$s \leftarrow s + e_{n_0-1}^T$
return s

(a)

Algorithm 5: ENCRYPT$^{\texttt{McE}}$

Input: $u = [u_0, \ldots, u_{n_0-2}]$: plaintext message; n_0 $1 \times p$ binary vectors; $e = [e_0, \ldots, e_{n_0-1}]$: error message with $wt(e)=t$; composed as n_0 $1 \times p$ binary vectors; $pk^{\texttt{McE}}=[Z \mid [M_0 \mid \ldots \mid M_{n_0-2}]^T]]$: public key; composed as sequence of n_0-1 $p\times p$ circulant blocks M_j, with $0\leq j<n_0-1$ juxtaposed to compose a systematic $(n_0-1)\times n_0$ generation matrix with $p\times p$ circulant blocks. Z: a diagonal block matrix with n_0-1 replicas of the identity circular block I

Output: $c = [c_0, \ldots, c_{n_0-1}]$: error affected codeword; $1 \times pn_0$ binary vector, where each c_j is a $1\times p$ vector with $0\leq j<n_0$.

Data: $p>2$ prime, $\texttt{ord}_p(2)=p-1$, $n_0 \geq 2$

$\texttt{blk} \leftarrow 0$ `// p × p zero block`
for $j = 0$ **to** $n_0 - 2$ **do**
    $\texttt{blk} \leftarrow \texttt{blk} + u_j^T\, M_j$
$c \leftarrow [u_0, u_1, \ldots, u_{n_0-2}, \texttt{blk}]$
for $j = 0$ **to** $n_0 - 1$ **do**
    $c_j \leftarrow c_j + e_j$
return c

(b)

Figure 1.2: Encryption algorithm of Niederreiter (a) and McEliece (b) cryptoschemes, instantiated with QC-LDPC codes

If we consider that the circulant blocks in any block row of Q have overall weight $m=\sum_{i=0}^{n_0-1} m_i$, the size of $sk^{\texttt{Nie}}$ (equiv., $sk^{\texttt{McE}}$) is $|sk^{\texttt{Nie}}|=n_0\,(d_v+m)\,\lceil \log_2(p)\rceil$ bits.

We note that, given the fast generation procedure for H and Q, a viable option to reduce the size of the private key *at rest* is to store the seed of a cryptographically secure Pseudo Random Number Generator (PRNG) from which H and Q are generated. To this end, we chose the NIST standard PRNG CTR-DRBG, instantiated with AES-256 as its core block cipher.

- The *encryption* algorithm ENCRYPT$^{\texttt{Nie}}$ in Fig. 1.2(a) takes as input a public key $pk^{\texttt{Nie}} = [M_0 \mid \ldots M_j \mid \ldots \mid M_{n_0-2}|I]$ and a message composed as a randomly chosen $1 \times pn_0$ binary vector e with exactly t asserted bits, and outputs a ciphertext that is the $p \times 1$ syndrome of the message computed multiplying sequence of n_0 message blocks of size $1 \times p$ by the QC

parity-check matrix of the public code included in the public key:

$$s = [M_0 \mid \dots M_j \mid \dots \mid M_{n_0-2}|I]e^T.$$

The *encryption* algorithm ENCRYPT$^{\texttt{McE}}$ in Fig. 1.2(b) takes three input parameters: a plaintext message composed as a $1 \times p(n_0 - 1)$ binary vector u, an *error vector* composed as a $1 \times pn_0$ binary vector e with exactly t asserted bits (supposed to be uniformly and randomly picked from the set of binary vectors with the same weight), and a public key $pk^{\texttt{McE}}$ structured as a QC generator matrix with size $(n_0 - 1)\times n_0$, i.e., $pk^{\texttt{McE}} = \left[Z \mid [M_0 \mid \dots \mid M_{n_0-2}]^T\right]$, where $Z = \texttt{diag}(I, \dots, I)$ is a diagonal block matrix composed as $n_0 - 1$ replicas of the identity circulant block I (i.e., a $p \times p$ identity matrix), which coincides with an $(n_0 - 1)p \times (n_0 - 1)p$ identity matrix. The algorithm outputs a ciphertext c that is an error affected $1 \times pn_0$ binary vector composed as follows. The sequence of the first (n_0-1) binary vectors from the plaintext message (each with size $1 \times p$) followed by the binary vector obtained as the sum of products between the corresponding $1 \times p$ blocks in u and the ones in the public key portion $[M_0 \mid \dots \mid M_{n_0-2}]$, is added to the sequence of n_0 binary vectors of the error vector e:

$$c = [e_0| \dots |e_{n_0-2}|e_{n_0-1}] + \left[u_0 \mid \dots \mid u_{n_0-2} \mid \sum_{j=0}^{n_0-2} u_j M_j\right].$$

- The *decryption* algorithm DECRYPT$^{\texttt{Nie}}$ in Fig. 1.3(a) takes as input a secret key $sk^{\texttt{Nie}} = \{H, Q\}$ and a ciphertext that is identified with a $p \times 1$ binary syndrome that is supposed to be computed as $s = [M_0 \mid \dots \mid M_{n_0-2}|I]e^T = \left(L_{n_0-1}^{-1}[L_0, \dots, L_{n_0-1}]\right) e^T$. The outcome of the decryption algorithm is the original binary vector e.

  The initial steps of the algorithm (lines 1-2) consist in taking the secret matrices, re-computing $L = HQ$ and executing the multiplication between the received syndrome s and the last block of $L = [L_0, \dots, L_{n_0-1}]$ to obtain a new $p\times 1$ binary vector $s' = L_{n_0-1}s = HQ\, e^T = H\left(Qe^T\right) = H\left(eQ^T\right)^T$.

  Defining the *expanded error vector* as $e' = eQ^T$, the binary vector s' can be thought of as $s' = He'^T$ to the end of applying a QC-LDPC decoding procedure and recover both e' and the original error vector $e = e'\left(Q^T\right)^{-1}$.
  QC-LDPC decoders are not bounded distance decoders, and some non-zero DFR must be tolerated. The system parameters can be chosen such that the DFR is acceptably small; for this purpose, the average decoding radius of the private code must be sufficiently larger than the Hamming weight of e', which is $t' \leq mt$ and approximately equal to mt, due to the sparsity of Q and e. As shown in line 3 of algorithm DECRYPT$^{\texttt{Nie}}$ in Fig. 1.3(a), by exploiting the efficient decoding algorithm described in Section 1.5 (named QDECODER) the computation of $\left(Q^T\right)^{-1}$ and the subsequent vector-matrix multiplication can be avoided altogether. In fact, the QDECODER described in Section 1.5 allows e to be recovered directly from the syndrome s, taking into account the effects of both H and Q separately. If the decoding procedure terminates successfully, the procedure returns also a Boolean variable $\texttt{res} = \texttt{true}$ along with the recovered original message e (see line 6 of algorithm DECRYPT$^{\texttt{Nie}}$ in Fig. 1.3(a)). On the other hand, when the decoding fails, the decryption algorithm returns the value of the Boolean variable `res` set to `false` and a null value for the original message $e = \perp$ (lines 4–5).

  The *decryption* algorithm DECRYPT$^{\texttt{McE}}$ in Fig. 1.3(b), takes as input a secret key $sk^{\texttt{McE}} = \{H, Q\}$ and a ciphertext that is identified with an error affected codeword $c = [c_0, \dots, c_{n_0-1}]$

Algorithm 6: DECRYPT$^{\texttt{Nie}}$

Input: s: syndrome; $1 \times p$ binary vector.
$sk^{\texttt{Nie}} = \{H, Q\}$ private key;
Output: $e = [e_0, \ldots, e_{n_0-1}]$: error; sequence of n_0 binary vectors with size $1 \times p$.
`res`: Boolean value denoting if the decryption ended successfully (`true`) or not (`false`)
Data: p>2 prime, $\texttt{ord}_p(2)$=$p-1$, $n_0 \geq 2$

1 $L \leftarrow HQ$ // $L = [L_0|\ldots|L_{n_0-1}]$
2 $s' \leftarrow L_{n_0-1}s$ // $s' = HQe^T$
3 $\{e, \texttt{res}\} \leftarrow$ QDECODER$(s', sk^{\texttt{Nie}})$
4 **if** `res` = `false` **then**
5 $e \leftarrow \perp$
6 **return** $(e, \texttt{res})$

(a)

Algorithm 7: DECRYPT$^{\texttt{McE}}$

Input: $c = [c_0, \ldots, c_{n_0-1}]$: error affected codeword; $1 \times pn_0$ binary vector, where each c_j is a $1 \times p$ vector with $0 \leq j < n_0$;
$sk^{\texttt{McE}} = \{H, Q\}$ private key;
Output: $u = [u_0, \ldots, u_{n_0-2}]$: message; sequence of n_0-1 binary vectors with size $1 \times p$.
$e = [e_0, \ldots, e_{n_0-1}]$: error; sequence of n_0 binary vectors with size $1 \times p$.
`res`: Boolean value denoting if the decryption ended successfully (`true`) or not (`false`).
Data: p>2 prime, $\texttt{ord}_p(2)$=$p-1$, $n_0 \geq 2$

1 $L \leftarrow HQ$ // $L = [L_0|\ldots|L_{n_0-1}]$
2 $s \leftarrow Lc^T$ // $p \times 1$ `binary syndrome`
3 $\{e, \texttt{res}\} \leftarrow$ QDECODER$(s, sk^{\texttt{McE}})$
4 **if** `res` = `true` **then**
5 **for** $j = 0$ **to** $n_0 - 1$ **do**
6 $u_j \leftarrow c_j + e_j$
7 **else**
8 $e \leftarrow \perp$; $u \leftarrow \perp$
9 **return** $(u, e, \texttt{res})$

(b)

Figure 1.3: Decryption algorithm of Niederreiter (a) and McEliece (b) cryptosystems, instantiated with QC-LDPC codes

that was computed as the element-wise addition between a $1 \times pn_0$ error vector with exactly t asserted bits and a $1 \times pn_0$ binary vector $[u_0, \ldots, u_{n_0-2}, \texttt{blk}]$, where $\texttt{blk} = \sum_{j=0}^{n_0-2} u_j(L_{n_0-1}HQ)$. The outcomes of the decryption algorithm are the original error vector e and message u.

The initial steps of the algorithm (lines 1-2) consist in taking the secret matrices, re-computing $L = HQ$ and executing the multiplication between the parity-check matrix L and the received error-affected codeword c to obtain a syndrome s as $p \times 1$ binary vector $s = Lc^T$.

On line 3 the efficient decoding algorithm described in Section 1.5 (named QDECODER) is employed to recover the original error vector e. If the decoding phase is successful (i.e., `res` = `true`), the original message is computed by adding element-wise the first $n_0 - 1$ binary blocks of the error affected codeword (i.e., the ciphertext) and of the recovered error vector (see lines 6–7 of algorithm DECRYPT$^{\texttt{McE}}$ in Fig. 1.3(b)); otherwise, the variable `res` is set to `false`, while the message and error vectors are set to null values, u=$\perp$, e=$\perp$ (see lines 8–9).

Algorithm 8: LEDAcrypt-KEM ENCAP

Input: $pk^{\texttt{Nie}}$: public key.
Output: c: encapsulated ephemeral key;
K: ephemeral key.
Data: p>2 prime, $\texttt{ord}_p(2)$=$p-1$, $n_0 \geq 2$;
$\text{ENCRYPT}^{\texttt{Nie}}(e, pk^{\texttt{Nie}})$: encryption function of the Niederreiter PKE;
$\mathcal{E}$: set of all possible binary error vectors e=$[e_0|\ldots|e_{n_0-1}]$, $wt(e)$=t

1 $e \xleftarrow{\$} \mathcal{E}$ `// uniform random picking`
2 $c \leftarrow \text{ENCRYPT}^{\texttt{Nie}}(e, pk^{\texttt{Nie}})$
3 $K \leftarrow \texttt{Hash}(e)$
4 **return** (c, K)

(a)

Algorithm 9: LEDAcrypt-KEM DECAPS

Input: $sk^{\texttt{Nie}}$: secret key
k: a secret random bitstring;
c: encapsulated key.
Output: K: decapsulated key.
Data: p>2 prime, $\texttt{ord}_p(2)$=$p-1$, $n_0 \geq 2$;
$\text{DECRYPT}^{\texttt{Nie}}(c, sk^{\texttt{Nie}})$: decryption function returning $\texttt{res} = \texttt{false}$ on an incorrect decoding, `true` and the original message e, otherwise.

1 $\{e, \texttt{res}\} \leftarrow \text{DECRYPT}^{\texttt{Nie}}(c, sk^{\texttt{Nie}})$
2 **if** $\texttt{res} = \texttt{true}$ **and** $wt(e) = t$ **then**
3 **return** $\texttt{Hash}(e)$
4 **else**
5 **return** $\texttt{Hash}(c|k)$

(b)

Figure 1.4: Description of the key encapsulation and decapsulation primitives of LEDAcrypt KEM

1.3 Description of LEDAcrypt KEM

In this section we describe the structure of the Key Encapsulation Method of LEDAcrypt, that is, LEDAcrypt KEM.

We employ the Niederreiter cryptosysems with OW-CPA described in the previous section together with an apt conversion to obtain a KEM with a twofold goal: provide a fast KEM with IND-CPA and ephemeral keys for low latency session establishment with perfect forward secrecy and a KEM with IND-CCA2 and long term keys for scenarios where key reuse may be desirable. We achieve this goal employing the same IND-CCA2 conversion applied to the QC-LDPC Niederreiter cryptosystem for both scenarios and achieving an appropriate DFR through code parameter tuning and, where needed, an additional IND-CCA2 redundant encryption technique.

In particular, we employ the $U_m^{\not\perp}$ construction defined in [19], which starts from a deterministic cryptosystem to build a KEM with IND-CCA2 in the Random Oracle Model (ROM) with a tight reduction.

The same construction was proven to achieve IND-CCA2 in the Quantum Random Oracle Model (QROM) in [20], with a tighter security reduction being reported in [21], starting from the assumption that the underlying deterministic cryptosystem exhibits OW-CPA, as it is the case with our Niederreiter PKC. The proofs in [19–21] take into account the possibility that the underlying cryptoscheme is characterized by a bounded correctness error δ. The instantiation of the $U_m^{\not\perp}$ construction employing the QC-LDPC code-based Niederreiter cryptoscheme and a cryptographically secure hash, $\texttt{Hash}(\cdot)$, is reported in Fig. 1.4. We chose, as the cryptographically secure hash to instantiate the $U_m^{\not\perp}$ construction, the NIST standard SHA-3 hash function.

As shown in algorithm LEDAcrypt-KEM Encap in Fig. 1.4(a), the encryption function of a Niederreiter cryptoscheme encapsulates (line 2) a random error vector with exactly t asserted bits (line 1) and derives the encapsulated key K as the hash of the error vector itself (line 3). The decapsulation procedure, shown in algorithm LEDAcrypt-KEM Decaps in Fig. 1.4(b), attempts to recover the error vector through an efficient syndrome decoding procedure employing the private key $sk^{\mathtt{Nie}} = \{H, Q\}$.

The $U_m^{\not\perp}$ construction as reported in [19–21] tests if the recovered error vector e is effectively the correct Niederreiter plaintext by re-encrypting it with the public key. In our instantiation of this construction it is possible to ascertain whether the outcome of a successful execution of the decryption function e is the correct plaintext via testing that its weight is exactly t (see line 2 in Fig. 1.4(b)). Such a test is justified by the fact that a unique syndrome exists for each error vector within the decoding radius of a QC-LDPC code. Such an observation saves us the cost of re-encrypting e.

In case of a decoding failure [19–21], the decapsulation procedure computes the returned outcome by hashing a secret value and the ciphertext (line 5). This prevents an adversary from distinguishing when a failure occurs due to malformed plaintext messages, i.e., messages with a number of asserted bits that is not exactly equal to t, from when a failure occurs due to the intrinsic behavior of the underlying QC-LDPC code. In other terms, the adversary cannot draw any conclusion about the decoding abilities of the code at hand when he/she is in control of composing messages that are not in the legitimate message space.

To provide IND-CCA2 for a given security level 2^λ, the authors of [19] state that it is required for the decryption function to have a correctness error $\delta \leq 2^{-\lambda}$. Given our goal of having both a fast and compact KEM with ephemeral keys and IND-CPA guarantees, as well as a KEM with IND-CCA2, we will provide different sets of parameters to be employed in the LEDAcrypt KEM construction, which are characterized by a DFR low enough to foil statistical attacks and achieve IND-CCA2 guarantees, while at the same time not hindering practical deployment.

1.4 Description of LEDAcrypt PKC

In this section we describe a construction to instantiate the LEDAcrypt PKC cryptoscheme providing a PKC with IND-CCA2 guarantees. While it is possible to employ LEDAcrypt KEM in a Key Encapsulation Module + Data Encapsulation Mechanism (KEM+DEM) combination with a symmetric encryption primitive, we note that such an approach may lead to a non-negligible ciphertext expansion in case plaintexts are small in size.

We provide a construction which starts from the QC-LDPC code-based McEliece cryptosystem described in the previous section, and derives a PKC exploiting the available capacity of the Public-Key Encryption (PKE) primitive to store the actual message content. It is worth noting that the systematic form of the generator matrix of the public code included in the public key $pk^{\mathtt{McE}}$ would easily allow any observer to recover the information word u embedded in an encrypted message c, without recovering the private key of the cipher (i.e., $sk^{\mathtt{McE}} = \{H, Q\}$). Nevertheless, the conversion proposed by Kobara and Imai in [25], with the purpose of maximizing the amount of message encrypted by a McEliece PKC, allows IND-CCA2 guarantees to be provided in the ROM. Therefore, the confidentiality of the information word as well as the security of the private key remain guaranteed by the hardness of the NP-hard general decoding problem even when a

systematic generator matrix is employed as public key.

In the following, we describe the basic encryption and decryption transformations and, subsequently, the mechanisms of the γ-conversion scheme [25] that allow us to obtain a IND-CCA2 version of LEDAcrypt PKC. As the Kobara-Imai (KI) γ-conversion is based on bit-wise manipulations, to provide a clear and detailed description of them we introduce some specific naming conventions. Bit-string values will be reported with a teletype font name (e.g., the plaintext to be encrypted will be denoted as `ptx`, and the resulting ciphertext will be denoted as `ctx`) while, the length of a bit-string, `s`, will also be expressed in bits and denoted as $l_{\texttt{s}}$.

1.4.1 Encryption and decryption transformations for IND-CCA2

The main intuition of the KI conversion relies on `xor`-combining a padded plaintext message with the output of a Deterministic Random Bit Generator (DRBG), seeded with the random bit-string extracted from a True Random Number Generator (TRNG).
To this end, we chose the NIST standard PRNG CTR-DRBG, instantiated with AES-256 as its core block cipher. Specifically, the original plaintext message (with size $l_{\texttt{ptx}}$ bits) is prefixed with a `constant` bit-string, with $l_{\texttt{const}}$ bits, and a bit-string named `lenField`, with $l_{\texttt{lenField}}$ bits, containing the value of $l_{\texttt{ptx}}$. This message is then concatenated with a sequence of zero or more null bits to align the overall length of such an *extended plaintext* to a byte-multiple for implementation convenience (i.e., $l_{\texttt{extendedPtx}} = 8 \cdot \left\lceil \frac{l_{\texttt{const}}+l_{\texttt{lenField}}+l_{\texttt{ptx}}}{8} \right\rceil$). The extended plaintext is then `xor`-combined with the output of a DRBG to allow the resulting *obfuscated plaintext* to be safely enciphered. This ensures the ciphertext to appear perfectly random.

In order for the IND-CCA2 property to hold, a fresh random value (i.e., a seed), encoded with $l_{\texttt{seed}}$ bits, should be available from a TRNG for each message encryption. To be able to successfully decrypt the plaintext, the seed of the DRBG is also enciphered alongside the message. The secret seed is `xor`-combined with the digest computed by a hash function fed with the obfuscated plaintext. The correctness of the resulting *obfuscated seed* is guaranteed by concatenating to the original seed value enough zero bits to match the length of the hash digest in case $l_{\texttt{seed}} < l_{\texttt{hash}}$, thus obtaining $l_{\texttt{obfuscatedSeed}} = l_{\texttt{hash}}$.

The extended plaintext bits are subsequently split into two bit-strings, namely `iword` and `leftOver`. The former, having size of $l_{\texttt{iword}}$ bits, is employed as the information word u to be encoded by the QC-LDPC code underlying the processing performed by the McEliece encryption function (i.e., $l_{\texttt{iword}} = p(n_0 - 1)$), while the latter (with size $l_{\texttt{leftOver}}$ bits) is concatenated with the output of the McEliece encryption function to yield the final ciphertext (`ctx`) of the KI-γ construction.

The obfuscated secret seed bits are employed as input to the constant-weight encoding function to obtain a binary error vector e, with exactly t asserted bits, which is in turn fed into the McEliece encryption function to complete a proper initialization of the encryption process. In case the number of bits of the obfuscated seed (i.e., $l_{\texttt{obfuscatedSeed}} = l_{\texttt{hash}}$) are less than the ones required by the constant-weight encoding procedure, random bits are extracted from the DRBG (or from a new random source) to match the requirements.
Currently, the most efficient definition of a constant-weight function (in terms of execution time – see next subsection) may fail to yield an output bit-string with exactly t asserted bits. Thus, the computation of the constant-weight function is repeated until the value of a proper error vector is returned.

Algorithm 10: LEDAcrypt PKC encryption transformation

Data: n, k, t: QC-LDPC code parameters. $n = pn_0$ codeword size, $k = p(n_0 - 1)$ information word size, t error correction capability, n_0 basic block length of the code, p circulant block size.
HASH: hash function with digest length in bits $l_{\texttt{hash}}$
$l_{\texttt{obfuscatedPtx}} = \max\left(p(n_0-1), \left\lceil \frac{l_{\texttt{const}}+l_{\texttt{ptx}}}{8} \right\rceil \cdot 8\right)$
$l_{\texttt{const}} = l_{\texttt{seed}}$
$l_{\texttt{iword}} = p(n_0 - 1)$
$l_{\texttt{eword}} = l_{\texttt{hash}}$

Input: `ptx`: plaintext bit-string, with $l_{\texttt{ptx}} \geq 0$
$pk^{\texttt{McE}}$: QC-LDPC based McEliece public key

Output: `ctx` ciphertext bit-string

1 `seed` $\leftarrow$ TRNG() `// bit-string with length` $l_{\texttt{seed}}$
2 `pad` $\leftarrow$ DRBG(`seed`) `// bit-string with length` $l_{\texttt{obfuscatedPtx}}$
3 `obfuscatedPtx` $\leftarrow$ ZEROPADBYTEALIGNED(`const`||`lenField`||`ptx`) $\oplus$ `pad`
4 `obfuscatedSeed` $\leftarrow$ ZEROEXTENDBYTEALIGNED(`seed`, $l_{\texttt{hash}}$) $\oplus$ HASH(`obfuscatedPtx`)
5 {`iword`, `leftOver`} $\leftarrow$ SPLIT(`obfuscatedPtx`, $l_{\texttt{iword}}$, $l_{\texttt{obfuscatedPtx}} - l_{\texttt{iword}}$)
6 $u \leftarrow$ TOVECTOR(`iword`) `//` $1 \times p(n_0 - 1)$ `information word vector`
7 **repeat**
8 $\{e,$ `encodingOk`$\} \leftarrow$ CONSTANTWEIGHTENCODER(`obfuscatedSeed`)
9 **until** `encodingOk` = `true`
10 $c \leftarrow \text{ENCRYPT}^{\texttt{McE}}(u, e, pk^{\texttt{McE}})$ `//` $1 \times pn_0$ `codeword`
11 `ctx` $\leftarrow$ TOBITSTRING(c)||`leftover` `// bit-string with` $l_{\texttt{ctx}} = pn_0$
12 **return** `ctx`

Figure 1.5: Description of the KI-γ encryption function adopted to define the encryption primitive of LEDAcrypt PKC

The final ciphertext is composed by a bit-string made of the binary elements of the codeword computed by the McEliece encryption function, concatenated with the `leftOver` bit-string.
In case the bit-length of `leftOver` is not a multiple of 8, `leftOver` is prefixed with a zero pad up to the nearest byte for the sake of implementation ease and efficiency.
It is worth noting that the KI-γ construction allows a plaintext message of arbitrary length to be encrypted. Employing the LEDAcrypt PKC construction, plaintext messages longer than $p(n_0 - 1) - l_{\texttt{const}} - l_{\texttt{lenField}}$ have a fixed ciphertext expansion of $p + l_{\texttt{const}} + l_{\texttt{lenField}}$.
In contrast a LEDAcrypt KEM+DEM approach requires a fixed ciphertext expansion of pn_0 bits.

A pseudo-code description of the KI-γ encryption algorithm is shown as Algorithm 10 in Fig. 1.5. The described procedure employs the QC-LDPC code parameters of choice (i.e., the ones of $\mathcal{C}(pn_0, p(n_0 - 1))$ that allows up to t bit errors to be corrected), and a hash function, HASH(), having a digest with a byte-aligned bit-length $l_{\texttt{hash}}$ as configuration parameters. We chose as our hash function to instantiate the construction the NIST Standard SHA-3 hash function.

Algorithm 11 in Fig. 1.6 reports the pseudocode of the LEDAcrypt PKC decryption procedure. The

Algorithm 11: LEDAcrypt PKC decryption transformation

Data: n, k, t: QC-LDPC code parameters. $n = pn_0$ codeword size, $k = p(n_0 - 1)$ information word size, t error correction capability, n_0 basic block length of the code, p circulant block size.
$\texttt{const} = 0^{l_{\texttt{seed}}}$
HASH: hash function with digest length in bits $l_{\texttt{hash}}$

Input: `ctx`: ciphertext bit-string.
$sk^{\texttt{McE}}$: LEDAcrypt PKC private key.

Output: `ptx` plaintext bit-string

$\texttt{cword}, \texttt{leftOver} \leftarrow \text{SPLIT}(\texttt{ctx}, pn_0, l_{\texttt{ctx}} - pn_0)$
$c \leftarrow \text{TOVECTOR}(\texttt{cword})$
$\{u, e, \texttt{res}\} \leftarrow \text{DECRYPT}^{\texttt{McE}}(c, sk^{\texttt{McE}})$
if $\texttt{res} = \texttt{true}$ and $wt(e) = t$ **then**
    $\texttt{iword} \leftarrow \text{TOBITSTRING}(u)$
    $\texttt{obfuscatedSeed} \leftarrow \text{CONSTANTWEIGHTDECODE}(e)$
    $\texttt{seed} \leftarrow \text{ZEROTRIM}(\texttt{obfuscatedSeed} \oplus \text{HASH}(\texttt{iword}||\texttt{leftOver}), l_{\texttt{seed}})$
    $\texttt{pad} \leftarrow \text{DRBG}(\texttt{seed})$
    $\texttt{extendedPtx} \leftarrow \texttt{obfuscatedPtx} \oplus \texttt{pad}$ // extendedPtx should equal const||lenField||ptx
    $\{\texttt{retrievedConst}, \texttt{ptx}\} \leftarrow \text{ZEROTRIMANDSPLIT}(\texttt{extendedPtx}, l_{\texttt{const}}, l_{\texttt{lenField}})$
    if $\texttt{retrievedConst} = \texttt{const}$ **then**
        return ptx
return $\perp$

Figure 1.6: Description of the KI-γ decryption function adopted to define the decryption primitive of LEDAcrypt PKC

algorithm performs, in reverse order, the steps dual to the ones in the encryption procedure. The value in the first $l_{\texttt{const}}$ bits of the retrieved extended plaintext message is compared with the fixed value of the constant considered in the KI-γ to reject the decryption outcome in case they mismatch. As the McEliece decryption transformation employs a QC-LDPC code with an intrinsically non-bounded decoding radius, the decryption process of the KI-γ may have an additional reason to fail. As shown in the pseudocode, the IND-CCA2 property (given that the decoding failure rate is low enough) is preserved making a decoding failure indistinguishable from an accidental error.

1.4.2 Constant weight encoding/decoding

A straightforward approach to implement the constant-weight encoding and decoding procedures may consist in applying the conversion mandated by a combinatorial number system [23], which maps (bijectively) any combination of n bits with t asserted bits to an integer value in the range $\{0, \ldots, \binom{n}{t}\}$ and viceversa thus, building an enumeration of the set of combinations. However, computing the said bijection requires a computational effort in the same range as computing the binomial $\binom{n}{t}$, which adversely impacts the performance of LEDAcrypt when n is in a range of tens of thousands and t in the hundreds range.

An alternative solution relies on tackling the problem of performing the constant-weight encoding

of a binary string considering it as the result of a very efficient encoding of a sequence of integers representing the length of the *zero runs* in the constant-weight string to be obtained. This approach yields practically exploitable results when the run-length encoding known as Golomb Coding [14] is exploited. Indeed, Golomb Coding encodes a sequence of binary symbols where one is far more frequent than the other (zeroes are far more frequent than ones in our constant-weight words) as a dense binary string with efficiencies which exceed 95% with the one-densities involved in LEDAcrypt [38]. The approach to encode the binary string is the following: a parameter d is estimated depending on the density of the zeroes in the string. In particular, given the probability of a symbol of the sequence being equal to zero, the value of the parameter d is derived as the integer rounding of the median of the distribution of the Bernoulli process modeling the occurrence of null bit as the successful event and of an asserted bit as an unsuccessful one. Once the value of d is determined, a run of zeroes of length l is encoded according to the following procedure:

i. Divide l by d, yielding a quotient q and a remainder r

ii. Encode q in unary, emitting q '1' symbols followed by a '0'

iii. Encode r through the *truncated binary encoding* [14], thus employing at most $\lceil \log_2(d) \rceil$ bits.

Given the high encoding efficiency of the said procedure, it is highly likely that a random binary string of appropriate length will decode to a constant weight string of weight t and length n if it is interpreted as the encoding of a set of t zero runs, with an appropriate estimate of the value of d.

The efficiency can be raised through recomputing the estimate of d after each run-length of 0s is encoded, as follows:

i. Compute the first estimate of d, said d_1, considering as the probability of a '1' occurring $p_1 = \frac{t_1}{n_1} = \frac{t}{n}$.

ii. Once the i-th length l_i of a run of zeroes is decoded, compute the new estimate of d, d_{i+1}, considering as the probability of a '1' occurring $p_{i+1} = \frac{t_i - 1}{n_i - l_i - 1}$.

This procedure yields an efficient way of obtaining a constant weight string, given an arbitrary binary string of length $\lfloor \log_2 \binom{n}{t} \rfloor$ performing its decoding onto a sequence of the length of zero runs present among the 1's of the constant weight word. Symmetrically, the constant weight error vector derived during the decryption procedure can be encoded into a compact binary string, yielding back the value of `eword` which was generated during the encryption.

1.5 Efficient decoding for LEDAcrypt primitives

While it is possible to perform decoding of the private QC-LDPC code in LEDAcrypt employing the parity-check matrix H and a classic BF decoder such as the one described in Algorithm 1, such a choice would not exploit to the utmost the efficiency and the correction power of the underlying QC-LDPC code. Indeed, given a syndrome s' obtained as $s' = HQe^T = H(eQ^T)^T$, the positions of the errors $e' = eQ^T$ to be corrected are not uniformly distributed; instead they depend on the positions of the set entries in Q^T, which are known to the decoder.

Starting from classical BF, we have developed an improved decoder that is specifically designed for LEDAcrypt, where the positions of the set entries in $e' = eQ^T$ to be corrected are influenced by the value of Q^T, as e' is equivalent to a random error vector e with weight t multiplied by Q^T. Since this improved decoder takes into account such a multiplication by the transpose of the matrix Q for estimating the locations of the bits to flip with greater efficiency, we denote it as *Q-decoder*.

The Q-decoding procedure employed in LEDAcrypt KEM is detailed in Algorithm 12 and attempts at reconstructing the secret error vector e from the received syndrome s. To this end, the peculiarity of the QDECODE algorithm is that it directly reconstructs the value of e, where a common bit-flipping decoder would retrieve $e' = eQ^T$, thus requiring a further matrix multiplication to complete the decryption action.

To perform the required syndrome decoding, QDECODE starts by computing the number of unsatisfied parity checks in the current syndrome in the same way a standard BF algorithm does (lines $5 - 9$ in Algorithm 12). Our approach to implement this computation is to exploit a sparse representation for the transposition of the parity check matrix H^T, which is taken as an input and denoted as `Htr` in the algorithm[1].

The differentiating point between the classical BF algorithm and the Q-decoding concerns how the bits of the codeword being decoded are selected for flipping. Indeed, while employing a classical BF algorithm would estimates set positions in e', on the base of the sole number of unsatisfied parity checks, the Q-decoder exploits the knowledge of the secret matrix Q^T to directly estimate set positions in e. This allows achieving important reductions in the number of decoding iterations and, as a further advantage, there is no need for computing and storing the inverse of the matrix Q^T, which would have been necessary in the case of the decoding procedure returning e'.

To this end, the Q-decoding computes, for each bit of e being decoded (lines 12–13 in Algorithm 12) a measure of similarity between the patterns of ones of a row of Q^T, blockwise cyclically shifted by the position of the bit of e itself, and the unsatisfied parity checks vector. If the similarity metric (lines 14–16 in Algorithm 12) is above a given threshold, both the error vector e and the value of the syndrome s for the next iteration `iter` are updated (lines 18–23 in Algorithm 12). The value of the aforementioned threshold can be obtained as a piecewise constant function of the current syndrome weight and the code parameters. For efficiency reasons, the function is precomputed and stored as a lookup table (`LutS`) containing pairs (weight, threshold). The Q-decoder computes the weight of the syndrome and determines the highest weight $\bar{\mathtt{w}}$ among the ones in the lookup table, which does not exceed the one of the syndrome (line 10 in Algorithm 12). The threshold for the similarity is selected as the one paired to $\bar{\mathtt{w}}$ in `LutS` (line 11 in Algorithm 12). Details on how the look-up table is obtained are given in Section 1.5.2. Inputs of the iterative decoding algorithm are the $p \times 1$ syndrome s', and the matrices H and Q composing the parity-check matrix $L = HQ$, while its status is composed by the current value of the syndrome and the current value of the estimated error vector.

The motivation of the use of a Q-decoder in LEDAcrypt is in the fact that it allows taking into account the particular geometry of the error vector e'. Indeed, the expanded error vector $e' = eQ^T$

[1]This, in turn, allows to reduce the number of iterations of the innermost loop of the parity check computation (lines 7–9 in Algorithm 12) from $\frac{n_0 p}{\mathtt{machine\_word}}$ to d_v. For example, considering the case of $n_0 = 2, p = 25931$ on the NIST reference platform ($\mathtt{machine\_word} = 64$) the number of iterations drops from 811 to 17.

Algorithm 12: Q-decoding

Input: s': QC-LDPC syndrome, binary vector of size p
`Htr`: transposed parity-check matrix, represented as an $n_0 \times d_v$ integer matrix containing the positions in $\{0, 1, \ldots, p-1\}$ of the set coefficients in the n_0 blocks of $H^T = [H_0^T \mid H_1^T \mid \ldots \mid H_{n_0-1}^T]$
`Qtr`: private matrix, represented as an $n_0 \times m$, $m = \sum_{i=0}^{n_0-1} m_i$ integer matrix containing the positions in $\{0, \ldots, n_0 p - 1\}$ of the asserted coefficients in Q^T rows

Output: e: the decoded error vector with size $n_0 p$
`decodeOk`: Boolean value denoting the successful outcome of the decoding action

Data: `imax`: the maximum number of allowed iterations before reporting a decoding failure
`LutS`: piecewise constant function yielding the value of the bit flipping threshold of similarity, given the syndrome weight.
It is represented as an array of (weight, threshold) pairs for all the boundary values of the piecewise function.

```
1  iter ← 0
2  repeat
3      unsat_pc ← [0 | ... | 0]    // array of n0p counters of unsatisfied parity checks
4      currSynd ← s'
5      for i = 0 to n0 − 1 do
6          for exp = 0 to p − 1 do
7              for h = 0 to dv − 1 do
8                  if GETBLOCKCOEFFICIENT(currSynd, (exp + Htr[i][h]) mod p) = 1 then
9                      unsat_pc[i · p + exp] ← unsat_pc[i · p + exp] + 1
10     w̄ ← MAX({w | (w, th) ∈ LutS ∧ w < WEIGHT(currSynd)})
11     th̄ ← th | (w̄, th) ∈ LutS
12     for i = 0 to n0 − 1 do
13         for exp = 0 to p − 1 do
14             similarity ← 0
15             for k = 0 to m − 1 do
                   // qrow contains the positions of the ones of a row of Q rotated intra-block by exp
16                 qrow[k] ← Qtr[i][k] − (Htr[i][k] mod p) + ((exp + Qtr[i][k]) mod p)
17                 similarity ← similarity + unsat_pc[qrow[k]]
18             if similarity ≥ th̄ then
19                 e[i · p + j] ← e[i · p + j] ⊕ 1
20                 for k = 0 to m − 1 do
21                     for h = 0 to dv − 1 do
22                         idx ← (Htr[qrow[k]/p][h] + (qrow[k] mod p)) mod p
23                         s'[idx] ← s'[idx] ⊕ 1
24     iter ← iter + 1
25 until s' ≠ 0  AND  iter < imax
26 if s' = 0 then
27     return e, true
28 return e, false
```

can be written as the sum of the rows of Q^T indexed by the support of e, that is

$$e' = \sum_{j \in \phi(e)} q_j \tag{1.15}$$

where $\phi(e)$ denotes the support of e. Since both Q and e are sparse (that is, $m, t \ll n$), cancellations between ones are very unlikely. Then, this fact can be exploited to improve the decoding procedure.

Indeed, first of all the vector *unsat_pc* expresses the likelihood of the bits in e' being set, in the sense that the larger the value of $unsat\_pc[i]$ is, the larger the probability that $e'_i = 1$. Then, the entries of *unsat_pc* are combined to obtain the likelihood values for set entries in e; indeed, let us look at the j-th bit in e, and consider that

- If $j \notin \phi(e)$, then it is very likely that q_j has a very small number of common ones with all the rows of Q^T forming e'. Hence the expected value of `similarity` is small.
- If $j \in \phi(e)$, then q_j is one of the rows of Q^T forming e', hence the expected value of `similarity` is large.

1.5.1 Q-decoder features

In this section we describe some of the features of the Q-decoder; we show that, when rejection sampling on the secret key is applied, the Q-decoder outcome is equivalent to that of a BF decoder applied on $L = HQ$, with some significant gain in the decoding complexity.

Lemma 1.5.1 (Equivalence of the bit-flipping decoder and Q-decoder) Let H and Q be the two matrices forming the parity-check matrix $L = HQ$, and denote as $\mathcal{L} \leftarrow \texttt{Lift}(L)$, $\mathcal{H} \leftarrow \texttt{Lift}(H)$, $\mathcal{Q} \leftarrow \texttt{Lift}(Q)$, the matrices obtained through lifting their values from $\mathbb{Z}_2$ to $\mathbb{Z}$. Assume a BF procedure acting on L and a Q-decoding procedure acting on $\mathcal{H}$ and $\mathcal{Q}$, both taking as input the same syndrome value s, providing as output an updated syndrome and a guessed error vector $\hat{e}$ (which is initialized as $\hat{e} = 0_{1\times n}$ at the beginning of the computations), and employing the same bit-flipping thresholds. If $\mathcal{L} = \mathcal{H}\mathcal{Q}$, the BF and Q-decoding procedures compute as output the same values for s and $\hat{e}$.

Proof. The functional equivalence can be proven showing that the update to the two state vectors, the syndrome s and the current $\hat{e}$ performed by the bit-flipping decoder and the Q-decoder leads to the same values at the end of each iteration of the decoding algorithms. We start by observing that the second phase of the BF and Q-decoder procedure will lead to the same state update of s and $\hat{e}$ if the values of the $\texttt{upc}^{(\mathrm{BF})}$ vector for the BF procedure and the $\texttt{upc}^{(\mathrm{Q-dec})}$ vector for the Q-decoder coincide. Indeed, since the update only depends on the values of the unsatisfied parity-checks and the flipping threshold b, if $\texttt{upc}^{(\mathrm{BF})} = \texttt{upc}^{(\mathrm{Q-dec})}$ the update on $\hat{e}$ and s will match. We consider, from now on, the parity-check computation procedures as described before through matrix multiplications over the integer domain, and prove that, during the first phase, the BF decoder and the Q-decoder yield values of $\texttt{upc}^{(\mathrm{BF})}$ and $\texttt{upc}^{(\mathrm{Q-dec})}$ such that $\texttt{upc}^{(\mathrm{BF})} = \texttt{upc}^{(\mathrm{Q-dec})}$ under the hypothesis that the starting values for s and $\hat{e}$ match. Considering the computation of $\texttt{upc}^{(\mathrm{BF})}$, and denoting with l_{ij} the element of L at row i, column j, we have that $\texttt{upc}^{(\mathrm{BF})} = \varsigma\,\mathcal{L}$, hence $\texttt{upc}_j^{(\mathrm{BF})} = \sum_{z=0}^{r-1} l_{zj}\, s_z$. The computation of $\texttt{upc}^{(\mathrm{Q-dec})}$ proceeds as follows $\texttt{upc}^{(\mathrm{Q-dec})} = (\varsigma\,\mathcal{H})\,\mathcal{Q} =$

$$\sum_{i=0}^{n-1}\left(\sum_{z=0}^{r-1} s_z\, h_{zi}\right) q_{ij} = \sum_{z=0}^{r-1}\left(\sum_{i=0}^{n-1} h_{zi}\, q_{ij}\right) s_z.$$

Recalling the hypothesis $\mathcal{L} = \mathcal{H}\mathcal{Q}$, it is possible to acknowledge that $\sum_{i=0}^{n-1} h_{zi}\, q_{ij} = l_{zj}$, which, in turn, implies that $\mathtt{upc}^{(\mathrm{Q-dec})} = \mathtt{upc}^{(\mathrm{BF})}$. ■

Lemma 1.5.2 (Computational advantage of the Q-decoder) Let us consider a bit-flipping decoding procedure and a Q-decoder procedure both acting on the same parity-check matrix $L = HQ$. The number of non-null entries of a column of H is $d_v \ll n$, the number of non-null entries of a column of Q is $m \ll n$, and the number of non-null entries of a column of L is $d_v m$ (assuming no cancellations occur in the multiplication HQ). The computational complexity of an iteration of the bit-flipping decoder equals $\mathcal{O}(d_v mn + n)$, while the computational complexity of an iteration of the Q-decoder procedure is $\mathcal{O}((d_v + m)n + n)$.

Proof. *(Sketch)* The proof can be obtained in a straightforward fashion with a counting argument on the number of operations performed during the iteration of the decoding procedures, assuming a sparse representation of H, L and Q. In particular, the amount of operations performed during the unsatisfied parity-check count estimation phase amounts to $\mathcal{O}(d_v mn)$ additions for the bit-flipping decoder and to $\mathcal{O}((d_v + m)n)$ for the Q-decoder, while both algorithms will perform the same amount of bit flips $\mathcal{O}(n + r) = \mathcal{O}(n)$ in the bit-flipping and syndrome update computations. ■

When $\mathcal{L} \neq \mathcal{H}\mathcal{Q}$, it is not possible to state the equivalence of BF decoding and Q-decoding according to Lemma 1.5.1. However, some qualitative considerations about their behavior can be drawn analyzing the product $\mathcal{H}\mathcal{Q}$ in the aforementioned case. Indeed, an entry in the i-th row, j-th column of $\mathcal{L}$ is different from the one with the same coordinates in $\mathcal{H}\mathcal{Q}$ whenever the scalar product i-th row of $\mathcal{H}$ and the j-th column of $\mathcal{Q}$ is ≥ 2. First of all, we note that such an event occurs with probability $\sum_{i=2}^{\min\{m,d_c\}} \frac{\binom{m}{i}\binom{n-m}{d_c-i}}{\binom{n}{d_c}}$, which becomes significantly small if the code parameters (n,d_c,m) take values of practical interest such as the ones reported in Chapter 3. Since the number of entries which have a different value in $\mathcal{H}\mathcal{Q}$ with respect to $\mathcal{L}$ are expected to be small, the values of the unsatisfied parity check counts $\mathtt{upc}^{(\mathrm{BF})}$ and $\mathtt{upc}^{(\mathrm{Q-dec})}$ are also expected to differ by a quite small amount, while the computational complexity of the decoding algorithms will remain substantially unchanged, as the term $d_v m$ in the BF decoder is reduced by a significantly small term, while the one of the Q-decoder is unchanged.

1.5.2 Choice of the Q-decoder decision thresholds

One important aspect affecting performance of Q-decoders is the choice of the threshold values against which the correlation is compared at each iteration. A natural choice is to set the threshold used at the l-th iteration equal to the largest entry in $\mathtt{unsatParityChecks}^{(\mathtt{Q-dec})}$. This strategy ensures that only those few bits that have maximum likelihood of being affected by errors are flipped during each iteration, thus achieving the lowest DFR. However, the strategy has some drawbacks in terms of complexity, since the computation of the maximum correlation entails significant memory storage and some repeated operations. For these reasons, we consider a different procedure, which allows computing the threshold values on the basis of the syndrome weight at each iteration. According to this approach, during an iteration it is sufficient to compute the syndrome weight and read the corresponding threshold value from a look-up table. The look-up table just depends on

the system parameters and can thus be precomputed: with this strategy the threshold selection at each iteration is really fast and, despite its very low complexity, still allows to achieve a sufficiently low DFR, within a significantly smaller number of decoding iterations, as our numerical simulations confirm.

Let us consider the l-th iteration of the Q-decoder, and denote as $s^{(l)}$ the input syndrome, corresponding to

$$s^{(l)} = H\left(e^{(l)}Q^T\right)^T = He'^{(l)T}, \tag{1.16}$$

where $e'^{(l)} = e^{(l)}Q^T$. We denote as t_l the weight of $e^{(l)}$ and as t'_l the weight of the corresponding expanded error vector $e'^{(l)}$; because of the sparsity of both e and Q, we assume that $t'_l = mt_l$. We introduce the following probabilities [5]

$$\begin{aligned} \mathtt{p}_{\mathtt{correct-unsatisfied}}(t'_l) &= \sum_{j=1,\, j \text{ odd}}^{\min\left[n_0 d_v - 1, t'_l\right]} \frac{\binom{n_0 d_v - 1}{j}\binom{n - n_0 d_v}{t'_l - j}}{\binom{n-1}{t'_l}} \\ \mathtt{p}_{\mathtt{incorrect-unsatisfied}}(t'_l) &= \sum_{j=0,\, j \text{ even}}^{\min\left[n_0 d_v - 1, t'_l - 1\right]} \frac{\binom{n_0 d_v - 1}{j}\binom{n - n_0 d_v}{t'_l - j - 1}}{\binom{n-1}{t'_l - 1}} \end{aligned} \tag{1.17}$$

where:

- $\mathtt{p}_{\mathtt{correct-unsatisfied}}(t'_l)$ is the probability that a codeword bit is error-free and a parity-check equation evaluates it wrongly, i.e., it is unsatisfied;
- $\mathtt{p}_{\mathtt{incorrect-unsatisfied}}(t'_l)$ is the probability that a codeword bit is in error and a parity-check equation evaluates it correctly, i.e., it is unsatisfied.

In both cases, the syndrome bit is equal to 1. So, the average syndrome weight at iteration l, which we denote as $w_s^{(l)}$, can be related to the sole value of t'_l as

$$w_s^{(l)} = p\left(\frac{t'_l}{n}\mathtt{p}_{\mathtt{incorrect-unsatisfied}}(t'_l) + \frac{n - t'_l}{n}\mathtt{p}_{\mathtt{correct-unsatisfied}}(t'_l)\right). \tag{1.18}$$

We point out that, since both the parity-check matrix and the error vector are sparse, the probability of any $\mathtt{wt}(s^{(l)})$ being significantly different from $w_s^{(l)}$ is negligible.

Therefore, eq. (1.18) allows predicting the average syndrome weight starting from t'_l. We now consider the i-th codeword bit and the corresponding entry in `unsatParityChecks`, which we denote as $\mathtt{upc}_i^{(\mathtt{Q-dec})}$. Let $\mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma \in [0; md_v]$: then, the probability that such a codeword bit

is affected by an error can be written as

$$Pr\left[e_i^{(l)} = 1 | \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right] = \frac{Pr\left[e_i^{(l)} = 1, \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right]}{Pr\left[\mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right]} =$$
$$= \frac{Pr\left[e_i^{(l)} = 1, \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right]}{Pr\left[e_i^{(l)} = 1, \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right] + Pr\left[e_i^{(l)} = 0, \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right]} =$$
$$= \left(1 + \frac{Pr\left[e_i^{(l)} = 0, \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right]}{Pr\left[e_i^{(l)} = 1, \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right]}\right)^{-1} \tag{1.19}$$

where $e_i^{(l)}$ is the i-th bit $e^{(l)}$. After some calculations, we obtain

$$Pr\left[e_i^{(l)} = 1 | \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right] = \frac{1}{1 + \frac{n-t_l}{t_l}\left(\frac{\mathtt{p}_{\mathtt{correct-unsatisfied}}(t_l)}{\mathtt{p}_{\mathtt{incorrect-unsatisfied}}(t_l)}\right)^{\sigma}\left(\frac{1-\mathtt{p}_{\mathtt{correct-unsatisfied}}(t_l)}{1-\mathtt{p}_{\mathtt{incorrect-unsatisfied}}(t_l)}\right)^{md_v-\sigma}} \tag{1.20}$$

where $\mathtt{p}_{\mathtt{correct-unsatisfied}}(t_l)$ and $\mathtt{p}_{\mathtt{incorrect-unsatisfied}}(t_l)$ are given in (1.17), with t_l as argument instead of t'_l.

Adding the i-th row of Q^T to the expanded error vector e' is the same as flipping the i-th bit of the error vector e. Hence, we can focus on e and on how its weight t_l changes during decoding iterations. The values of t_l can be estimated as t'_l/m, due to the sparsity, while those of t'_l can be estimated according to (1.18).

We now want to define values of $\mathtt{upc}_i^{(\mathtt{Q-dec})}$ for which the decision to flip the i-th codeword bit is sufficiently reliable; this condition can be expressed as

$$Pr\left[e_i^{(l)} = 1 | \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right] > (1+\Delta) Pr\left[e_i^{(l)} = 0 | \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right] \tag{1.21}$$

where $\Delta \geq 0$ represents a margin that must be chosen taking into account the DFR and complexity: increasing Δ decreases the DFR but increases the number of decoding iterations. So, a trade-off value of Δ can be found that allows achieving a low DFR while avoiding unnecessary large numbers of iterations.

Since $Pr\left[e_i^{(l)} = 0 | \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right] = 1 - Pr\left[e_i^{(l)} = 1 | \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right]$, (1.21) can be rewritten as

$$Pr\left[e_i^{(l)} = 1 | \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right] > \frac{1+\Delta}{2+\Delta}. \tag{1.22}$$

$Pr\left[e_i^{(l)} = 1 | \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right]$ is an increasing function of σ, hence the minimum value of σ such that (1.22) is satisfied can be computed as

$$b^{(l)} = \min\left\{\sigma \in [0; md_v] \ : \ Pr\left[e_i = 1 | \mathtt{upc}_i^{(\mathtt{Q-dec})} = \sigma\right] > \frac{1+\Delta}{2+\Delta}\right\} \tag{1.23}$$

and used as the decision threshold at iteration l.

Based on the above considerations, the procedure to compute the decision threshold value per each iteration as a function of the syndrome weight can be summarized as follows:

i. The syndrome weights corresponding to $t'_l = 0, m, 2m, \cdots, mt$ (which are all the possible values of t'_l neglecting cancellations) are computed according to (1.18). These values are denoted as $\{w_s(0), w_s(m), \cdots, w_s(mt)\}$.

ii. At iteration l, given the syndrome weight $\bar{w_s}^{(l)}$, the integer $j \in [0, t]$ such that $w_s(jm)$ is as close as possible to $\bar{w_s}^{(l)}$ is computed.

iii. Consider $t_l = j$ and compute $b^{(l)}$ according to (1.23) using (1.20). The value of $b^{(l)}$, so obtained, is used as the decoding threshold for iteration l.

The above procedure can be implemented efficiently by populating a look-up table with the pairs $\{w_j, b_j\}$, sequentially ordered. During an iteration, it is enough to compute $\bar{w_s}^{(l)}$, search the largest w_j in the look-up table such that $w_j < \bar{w_s}^{(l)}$ and set $b^{(l)} = b_j$.

We have observed that, moving from the largest values of w_j to the smallest ones, the threshold values computed this way firstly exhibit a decreasing trend, then start to increase. According to numerical simulations, neglecting the final increase is beneficial from the performance standpoint. Therefore, in the look-up table we replace the threshold values after the minimum with a constant value equal to the minimum itself.

1.6 Analysis of the decryption failure rate

In this section we provide a theoretical analysis of the DFR of the Q-decoder used in LEDAcrypt instances with long term keys, adopting two iterations in which the thresholds are suitably chosen. In particular, the threshold of the second iteration is obtained through an analysis of each specific key pair, in order to ensure that the decoder can correct all error patterns whose weight does not exceed some value $\bar{t}$. The threshold of the first iteration is then optimized in order to guarantee that the number of residual errors is within the error correction capability of the second decoder iteration with sufficiently high probability. We first find the conditions under which the second iteration is able to correct all residual bit errors; then we study the first iteration in probabilistic terms, and compute the probability that the number of its residual bit errors is such that the second iteration achieves complete correction (i.e., the number of residual errors is $\leq \bar{t}$).

1.6.1 Null DFR for a single Q-decoder iteration

We consider a single iteration of the Q-decoder, applied on a syndrome s corresponding to an error vector e with weight t. To characterize the correction capabilities of the Q-decoder, we consider the case where $L = HQ$ has no cancellations, as enforced by our rejection sampling of the keys. Under this hypothesis, it is possible to use Lemma 1.5.1, and analyze the behavior of the corresponding bit flipping decoder acting on L, with the same bit flipping thresholds of the Q decoder. Let b denote the decoding threshold; ς_i denotes the i-th entry in ς and the element in the i-th row and j-th column in $\mathcal{L}$ is denoted as $l_{i,j}$. The number of unsatisfied parity checks of the i-th bit is denoted as $\mathtt{upc}_i$, and is computed as

$$\mathtt{upc}_i = \sum_{j=0}^{r-1} \varsigma_j \, l_{j,i} = \sum_{j=0}^{r-1} l_{j,i} \left(\bigoplus_{k=0}^{n-1} e_k l_{j,k} \right) = \sum_{\substack{j \in [0; r-1] \\ l_{j,i}=1}} \left(\bigoplus_{k=0}^{n-1} e_k l_{j,k} \right).$$

From now on, we will denote the weight of a column of L as $v = md_v$; we thus have that, for all the possible error locations $i \in \{0, \ldots, n-1\}$, $\mathtt{upc}_i \in \{0, \ldots, v\}$. We will also denote with l_i the i-th column of l, and with $\phi(l_i)$ the set of the positions of the non null elements in l_i. Let $L_{(S)}$ denote the matrix composed by the rows of L indexed by a set of integers S having elements $s \in S$ such that $s \in \{0, \ldots, p-1\}$ and $l_{(S),i}$ denote its $i-th$ column. We thus have that $L_{(\phi(l_i))}$ is a rectangular matrix with v rows of n elements, where the i-th column, $l_{(\phi(l_i)),i}$ is constituted by all ones.

Equation (1.24) can be thus rewritten as:

$$\mathtt{upc}_i = \mathrm{wt}\left(\bigoplus_{j \in \phi(e_j)} l_{(\phi(l_i)),j}\right) = \mathrm{wt}\left(L_{(\phi(l_i))} e^T\right). \tag{1.24}$$

Recall that the bit flipping decoder will change the i-th bit of the initially null error estimate $\hat{e}$ if and only if $\mathtt{upc}_i$ is greater or equal than threshold b. Depending on whether the correct value for the error bit is $e_i = 0$ or $e_i = 1$, the decoder will make a correct choice at the first iteration if it does not flip the i-th bit, or if it does, respectively. We now consider a worst case analysis for the capability of the decoder to make the correct flipping choice. In the following, we will denote the support of the vector e as $\phi(e)$.

If $e_i = 0$, the decoder will make a correct decision if $\mathtt{upc}_i < b$: we thus consider the following upper bound for $\mathtt{upc}_i$, assuming $e_i = 0$:

$$\mathtt{upc}_i = \mathrm{wt}\left(L_{(\phi(l_i))} e^T\right) = \mathrm{wt}\left(\bigoplus_{j \in \phi(e_j)} l_{(\phi(l_i)),j}\right) \leq \sum_{j \in \phi(e)} \mathrm{wt}\left(l_{(\phi(l_i)),j}\right). \tag{1.25}$$

By contrast, if $e_i = 1$ the decoder will make a correct decision if $\mathtt{upc}_i \geq b$. We thus consider the following lower bound for $\mathtt{upc}_i$ assuming $e_i = 1$:

$$\begin{aligned} \mathtt{upc}_i &= \mathrm{wt}\left(\bigoplus_{j \in \phi(e_j)} l_{(\phi(l_i)),j}\right) = \mathrm{wt}\left(l_{(\phi(l_i)),i} \oplus \bigoplus_{j \in \phi(e_j) \setminus i} l_{(\phi(l_i)),j}\right) \\ &= v - \mathrm{wt}\left(\bigoplus_{j \in \phi(e_j) \setminus i} l_{(\phi(l_i)),j}\right) \geq v - \sum_{j \in \phi(e_j) \setminus i} \mathrm{wt}\left(l_{(\phi(l_i)),j}\right). \end{aligned} \tag{1.26}$$

A relevant quantity in Equation (1.26) is the Hamming weight of the columns $l_{(\phi(l_i)),j}$ where $i \neq j$. Indeed, it represents the amount of interference on the $\mathtt{upc}_i$ value induced by the presence of errors in positions different from i.

To ease the computation of $wt(l_{(\phi(l_i)),j})$, we observe that it is equivalent to $wt\left(\phi(l_i) \cap \phi(l_j)\right)$, since $L_{(\phi(l_i))}$ is constituted of the rows of L where the column l_i contains a one, and only the ones present in such rows of the j-th column of L will contribute to $wt\left(l_{(\phi(l_i)),j}\right)$. We therefore define Γ as a $n \times n$ positive integer matrix for which its i-th row, j-th column entry , $\gamma_{i,j}$, is $wt\left(\phi(l_i) \cap \phi(l_j)\right)$. We can rewrite, after this definition, Equation (1.25) and Equation (1.26) as:

$$\mathtt{upc}_i \leq \sum_{j \in \phi(e)} \gamma_{i,j} \quad \text{and} \quad \mathtt{upc}_i \geq v - \sum_{j \in \phi(e_j) \setminus i} \gamma_{i,j} \tag{1.27}$$

respectively.

In particular, we here consider the case in which the number of unsatisfied parity-check equations, for the case of an error affected bit, is always larger than that of an error free bit; in such a case, the decoder can correct all the input error patterns. In other words, it must be

$$\sum_{j\in\phi(e)} \gamma_{i,j} < v - \sum_{j\in\phi(e_j)\setminus i} \gamma_{i,j}, \quad \forall e \in \mathbb{F}_2^n, \quad wt\,(e) = t. \tag{1.28}$$

Let

$$\mu(z) = \max_{\substack{i\in\mathbf{N}, e\in\mathbb{F}_2^n \\ wt(e)=z,\ e_i=0}} \left\{ \sum_{j\in\phi(e)} \gamma_{i,j} \right\}. \tag{1.29}$$

For the case of $e_i = 0$ we have

$$\mathtt{upc}_i \leq \mu(t), \tag{1.30}$$

while, for $e_i = 1$, it is

$$\mathtt{upc}_i \geq v - \mu(t-1). \tag{1.31}$$

Equation (1.28) can then be rewritten as

$$\mu(t) + \mu(t-1) < v. \tag{1.32}$$

Let $\bar{t}$ be the largest integer for which condition (1.32) is satisfied; then, if the decoding threshold is in $[\mu(t)+1; v-\mu(t)]$, the decoder will correct all error vector of weight $t \leq \bar{t}$.

1.6.2 Efficient computation of $\bar{t}$

We will now prove that Γ is quasi cyclic. Let l_i be the i-th column of $\mathcal{L}$. Note that Γ is a symmetric matrix, since it holds that $\gamma_{i,j} = \gamma_{j,i}$, for all pairs of indexes i, j due to the symmetry of the intersection operation. Consider the indexes $i, j \in \{0, \ldots, n_0 p - 1\}$ as obtained as $i = pi_p + i'$ and $j = pj_p + j'$, respectively, with $i_p, j_p \in \{0, \ldots n_0 - 1\}$ and $i', j' \in \{0, \ldots, p-1\}$. Denoting with P the $p \times p$ circulant permutation matrix associated to the cyclic shift of one position, and observing that $P^p = I_p$, and that $(P^a)^T = P^{p-a}$, we have that

$$l_i = P^{i'} h_{i_p}, \qquad l_j = P^{j'} h_{j_p} \tag{1.33}$$

From this observation, we can derive

$$\begin{aligned}
\gamma_{i,j} = l_i^T\, l_j &= \left(P^{i'}\, l_{i_p}\right)^T P^{j'}\, l_{j_p} \\
&= l_{i_p}^T\, P^{p-i'} P^{j'}\, l_{j_p} = l_{i_p}^T\, P^{p+j'-i'}\, l_{j_p} \\
&= \begin{cases} l_{i_p}^T\, P^{j'-i'}\, l_{j_p} & \text{if } j' > i' \\ l_{i_p}^T\, P^{i'-j'}\, l_{j_p} & \text{if } j' < i' \end{cases} \\
&= \begin{cases} l_{i_p}^T\, l_{pj_p+j'-i'} & \text{if } j' > i' \\ l_{i_p}^T\, l_{pj_p+p-(i'-j')} & \text{if } j' < i' \end{cases} \\
&= \begin{cases} \gamma_{pi_p, pj_p+j'-i'} & \text{if } j' > i' \\ \gamma_{pi_p, pj_p+i'-j'} & \text{if } j' < i' \end{cases} \\
&= \gamma_{pi_p, pj_p+(j'-i' \mod p)},
\end{aligned} \tag{1.34}$$

which proves that matrix Γ is quasi cyclic with block size p, and can thus be written as:

$$\Gamma = \begin{bmatrix} \Gamma_{0,0} & \Gamma_{0,1} & \cdots & \Gamma_{0,n_0-1} \\ \Gamma_{1,0} & \Gamma_{1,1} & \cdots & \Gamma_{1,n_0-1} \\ \vdots & \vdots & \ddots & \vdots \\ \Gamma_{n_0-1,0} & \Gamma_{n_0-1,1} & \cdots & \Gamma_{n_0-1,n_0-1} \end{bmatrix}, \tag{1.35}$$

where each $\Gamma^{(i,j)}$ is a $p \times p$ circulant integer matrix; in particular, due to symmetry, we also have that $\Gamma_{i,j}^T = \Gamma_{j,i}$.

Because of these properties, it can be easily shown that the rows of Γ are not fully independent among themselves; in particular, this means that the whole matrix is described by a subset of its elements (actually, only $\frac{p-1}{2} n_0(n_0+1) - n_0$ entries are needed).

Essentially, this property can be used to obtain an efficient method for computing the value of $\bar{t}$ which, according to Equation (1.32), only depends on the values of $\mu(t)$. In the following we describe how, for the case of LEDAcrypt systems, the computation of $\mu(t)$ can be efficiently made.

First of all, the following procedure can be used to compute all the independent entries of Γ:

i. choose $i \in \{0, 1, \cdots, n_0 - 2\}$;

ii. choose $j \in \{i, \cdots, n_0 - 1\}$;

iii. define $\phi_i = \left\{a_0^{(i)}, \cdots, a_{md_v-1}^{(i)}\right\}$ and $\phi_j = \left\{a_0^{(j)}, \cdots, a_{md_v-1}^{(j)}\right\}$ as the supports of the first column of L_i and L_j, respectively;

iv. define $\psi_{i,j}$ as the length-p vector whose z-th entry is computed as $\left|\phi_i \cap \phi'_j\right|$, with

$$\phi'_j = \{\, a + z \mod p \,|\, a \in \phi_j \}\,.$$

If $i = j$, the first entry of $\psi_{i,j}$ is set as 0.

Once the vectors $\psi_{i,j}$ have been obtained, the value of $\mu(t)$ can be computed as follows:

i. start with $\mu(t) = 0$, and consider all values $i \in [0; n_0 - 1]$;

ii. for each value of i, concatenate the vectors $\psi_{i,j}$, with $j \in [i; n_0 - 1]$, and vectors $\psi_{j,i}$, with $j \in [0; i-1]$; sort the elements of the so obtained vector in descending order and consider the sum of its first t elements: if this sum is larger than $\mu(t)$, update $\mu(t)$.

1.6.3 Probabilistic analysis of the first iteration of the Q-decoder

In this section we provide a probabilistic analysis for the first iteration of the Q-decoder, with the aim of obtaining the probability that the decoder can, in the first iteration, correct a sufficiently large amount of errors. We consider the decoding procedure employed by the LEDA cryptosystems assuming that Lemma 1 holds. Given the equivalence of the BF decoder and Q-decoder provided by this Lemma, for the sake of simplicity, we will reason on the application of one iteration of the

BF decoder taking as input the $r \times n$ parity-check matrix $L = HQ$, assumed to be computed as a cancellation-free product between H and Q. Thus, L corresponds to the parity-check matrix of a regular LDPC code, with constant row-weight $d' - c = n_0 m d_v$ and constant column weight $d'_v = m d_v$. We denote the input syndrome as $s = Le^T$, where e is a length-n vector with weight t, and consider an initially null guessed error vector $\hat{e} = 0_{1\times n}$. We denote with b the employed decoding threshold, and with $\mathtt{upc}_j$ the number of unsatisfied parity-check equations in which the j-th bit participates; the decoder will flip the j-th position, i.e., will set $\hat{e}_j = 1$, if and only if $\mathtt{upc}_j \geq b$. Then, the second iteration will take as input the syndrome obtained as $s + L\hat{e}^T$, corresponding to the error vector $e \oplus \hat{e}$.

In the following, we model the number of residual errors which are left by the first decoder iteration, which corresponds to $wt\,(e \oplus \hat{e})$, i.e., to the number of differences between the actual error vector e and the guessed $\hat{e}$. We define $\mathtt{T}$ as a random variable over the discrete domain of integers $\{0, \ldots, n\}$, $t \geq 0$ having a probability mass function $Pr\,[\mathtt{T} = \tau]$, $\tau = wt\,(\hat{e} \oplus e)$ depending on the decoding strategy and the code parameters.

Let us consider a position j, and take into account the fact that a difference exists when either event $\{\hat{e}_j = 0, e_j = 1\}$ of $\{\hat{e}_j = 1, e_j = 0\}$ happens. To quantify such probabilities, we recall the probabilities $\mathtt{p_{correct-unsatisfied}}$ and $\mathtt{p_{incorrect-unsatisfied}}$ already introduced in Section 1.5.2.

Let $\mathtt{p_{correct}}$ be the probability of occurrence of the event $\{\hat{e}_j = 1 | e_j = 1\}$, i.e., the probability that the decoder flips (i.e., sets $\hat{e}_j = 1$) a bit in a position corresponding to a set entry in the actual error vector e. Such a probability can be obtained as

$$\mathtt{p_{correct}} = \sum_{j=b}^{d'_v} \binom{d'_v}{j} \left(\mathtt{p_{incorrect-unsatisfied}}(t)\right)^j \left(1 - \mathtt{p_{incorrect-unsatisfied}}(t)\right)^{d'_v - j}. \tag{1.36}$$

Analogously, we define $\mathtt{p_{induce}}$ as the probability that event $\{\hat{e} = 1 | ej = 0\}$ occurs, that is, the probability that the decoder flips a bit in a position j which corresponds to a null entry in the actual error vector e. Such a probability can be obtained as

$$\mathtt{p_{induce}} = \sum_{j=b}^{d'_v} \binom{d'_v}{j} \left(\mathtt{p_{correct-unsatisfied}}(t)\right)^j \left(1 - \mathtt{p_{correct-unsatisfied}}(t)\right)^{d'_v - j}. \tag{1.37}$$

Note that $\mathtt{p_{correct}}$ is indeed the probability that the Q-decoder performs a correct flip at the first iteration, while $\mathtt{p_{induce}}$ is the one of performing a wrong flip. By assuming that the decisions on the bits being flipped or not are taken independently, we have

$$Pr\left[f_{\mathtt{correct}} = c, f_{\mathtt{wrong}} = w\right] = Pr\left[f_{\mathtt{correct}} = c\right] \cdot Pr\left[f_{\mathtt{wrong}} = w\right],, \tag{1.38}$$

such that the probabilities of the Q-decoder performing $c \in \{0, \ldots, t\}$ correct flips out of t or $w \in \{0, \ldots, n - t\}$ wrong flips out of $n - t$ can be quantified introducing the random variables $f_{\mathtt{correct}}$ and $f_{\mathtt{wrong}}$, as follows

$$\begin{aligned} Pr\left[f_{\mathtt{correct}} = c\right] &= \binom{t}{c} {\mathtt{p_{correct}}}^c \left(1 - \mathtt{p_{correct}}\right)^{t-c}, \\ Pr\left[f_{\mathtt{wrong}} = w\right] &= \binom{n-t}{w} {\mathtt{p_{induce}}}^w \left(1 - \mathtt{p_{induce}}\right)^{n-t-w}. \end{aligned} \tag{1.39}$$

It is easy to see that, in the case of $f_{\texttt{correct}} = c$ and $f_{\texttt{wrong}} = w$, we have $wt\,(e + \hat{e}) = t + w - c$: thus, the probability that the guessed error vector $\hat{e}$, at the end of the computation of the first iteration of the Q-decoder, differs from the actual error vector in $\tau \in \{0, \ldots, t\}$ positions can be obtained as follows

$$Pr\left[\mathtt{T} = \tau\right] = \sum_{i=t-\tau}^{t} Pr\left[f_{\texttt{correct}} = i\right] \cdot Pr\left[f_{\texttt{wrong}} = \tau + i - t\right]. \tag{1.40}$$

This result permits us to estimate the probability of having a given number of errors $\tau \in \{0, \ldots, n\}$ left to be corrected after the first iteration of the Q-decoder, since in this case the hypothesis on the independence of the decisions to flip or not to flip a given variable can be assumed safely.

1.6.4 DFR characterization for a two-iteration Q-decoder

In this section we describe how the results obtained in the previous sections can be combined, in order to derive a theoretical characterization of the DFR. We consider a Q-decoder performing two iterations, and denote with b_0 and b_1 the decoding thresholds which are employed in the first and second iteration, respectively. In particular, according to the analysis provided in section 1.6.1, we suppose that the threshold b_1 has been chosen such that the decoder can correct all error patterns whose weight is not greater than some integer $\bar{t}$. Then, all initial error vectors of weight t which result, after the first decoder iteration, in a number of residual errors that is $\leq \bar{t}$, can be corrected by the decoder. This means that the DFR of this two iterations Q-decoder can be overestimated by the probability that the first iteration leaves more than $\bar{t}$ errors to be corrected. By means of Equation (1.40), such a probability can be estimated as

$$Pr\left[\mathtt{T} > \bar{t}\right] = 1 - \sum_{\tau=0}^{\bar{t}} Pr\left[\mathtt{T} = \tau\right]. \tag{1.41}$$

We have used this criterion to devise parameters sets with properly low DFR values, in order to allow for the use of long term keys.

Chapter 2

Security analysis of LEDAcrypt

In order to analyze the security of the LEDAcrypt primitives, we start by providing a quantitative assessment of NIST's security level targets in terms of the number of classical and quantum elementary operations required to perform an exhaustive key search against AES. Once the required security levels are delineated we analyze the attacks against LEDAcrypt highlighting their computational complexity, with the goal of providing an automated parameter design procedure for LEDAcrypt cryptosystems.

2.1 Security level goals

The bar to be cleared to design parameters for post-quantum cryptosystems is the computational effort required on either a classical or a quantum computer to break the AES with a key size of λ bits, $\lambda \in \{128, 192, 256\}$, through an exhaustive key search. The three pairs of computational efforts required on a classical and quantum computer correspond to NIST Category 1, 3, and 5, respectively [32]. Throughout the design of the parameters for the LEDA cryptosystems we ignore Categories 2 and 4: if a cipher matching those security levels is required, we advise to employ the parameters for Categories 3 and 5, respectively.

The computational worst-case complexity of breaking AES on a classical computer can be estimated as $2^{\lambda}\mathtt{C}_{\mathtt{AES}}$, where $\mathtt{C}_{\mathtt{AES}}$ is the amount of binary operations required to compute AES on a classical computer on a small set of plaintexts, and match them with a small set of corresponding ciphertexts to validate the correct key retrieval. Indeed, more than a single plaintext-ciphertext pair is required to retrieve AES keys [15]. In particular, a validation on three plaintext-ciphertext pairs should be performed for AES-128, on four pairs for AES-192 and on five for AES-256.

Willing to consider a realistic AES implementation for exhaustive key search purposes, we refer to [43], where the authors survey the state-of-the-art of Application-Specific Integrated Circuit (ASIC) AES implementations, employing the throughput per Gate Equivalent (GE) as their figure of merit. The most performing AES implementations are the ones proposed in [43], and require around 16ki GEs. We thus deem reasonable to estimate the computational complexity of an execution of AES as 16ki binary operations. We are aware of the fact that this is still a conservative estimate, as we ignore the cost of the interconnections required to carry the required data to the AES cores.

Table 2.1: Classical and quantum computational costs to perform an exhaustive key search on AES

NIST Category	AES Key Size (bits)	Classical Cost (binary operations)	Quantum Cost [15] (quantum gates)
1	128	$2^{128} \cdot 2^{14} \cdot 3 = 2^{143.5}$	$1.16 \cdot 2^{81}$
3	192	$2^{192} \cdot 2^{14} \cdot 4 = 2^{208}$	$1.33 \cdot 2^{113}$
5	256	$2^{256} \cdot 2^{14} \cdot 5 = 2^{272.3}$	$1.57 \cdot 2^{145}$

The computational complexity of performing an AES key retrieval employing a quantum computer was measured first in [15], where a detailed implementation of an AES breaker is provided. The computation considers an implementation of Grover's algorithm [16] seeking the zeros of the function given by the binary comparison of a set of AES ciphertexts with the encryption of their corresponding plaintexts for all the possible key values. The authors of [15] chose to report the complexity of the quantum circuit computing AES counting only the number of the Clifford and T gates. Selecting a different choice for the set of quantum gates employed to realize the AES circuit may yield a different complexity; however, the difference will amount to a reasonably small constant factor, as it is possible to re-implement the Clifford and T gates at a constant cost with any computationally complete set of quantum gates. We thus consider the figures reported in [15] as a reference for our parameter design procedure. In Table 2.1 we summarize the computational cost of performing exhaustive key searches on all three AES variants (i.e., with 128, 192, and 256 bits long keys), both considering classical and quantum computers. For the sake of simplicity, in the following we will round up fractional exponents of the reported complexities to the next integer value.

2.2 Hardness of the underlying problem

The set of computational decision problems for which an efficient solution algorithm can be devised for a non-deterministic Turing Machine (TM) represents a fruitful computational class from which primitives for asymmetric cryptosystems have been designed. Such a computational class, known as the Nondeterministic-Polynomial (NP) class, is characterized by problems for which it is efficient (i.e., there is a polynomial-time algorithm) to verify the correctness of a solution on a deterministic TM, while finding a solution to the problem does not have in general an efficient algorithm on a deterministic machine, hence the computational asymmetry required to build a cryptosystem.

When considering a quantum TM, i.e., the abstract computational model for a quantum computer, the class of problems which can be solved in polynomial time, with the quantum TM providing the correct answer with probability $> \frac{2}{3}$, is known as the Bounded-error Quantum Polynomial (BQP) time class [8].

In 1997 Peter Shor proved that the integer factoring problem, which has its decisional version in NP, is effectively in BQP [40], in turn demonstrating that a widely adopted cryptographic trapdoor function can be broken in polynomial time by a quantum computer. Consequentially, to devise a proper post-quantum asymmetric primitive it is crucial to choose a computational problem which resides outside BQP as its underlying foundation. While there is no current formal proof, a sub-

class of NP, the NP-complete problem class, is widely believed to contain computational problems not belonging to BQP, thus allowing only a polynomial speedup in their solution with a quantum TM.

LEDAcrypt is constructed starting from the computational search problems of either performing the decoding of a codeword (i.e., finding the error vector affecting a codeword) given a generic random linear code, or decoding a syndrome (i.e., finding the unique error vector corresponding to it) via a generic random parity-check matrix. The decision problems corresponding to the aforementioned problems were shown to be NP-complete in [7, 34]. As there is currently no search to decision reduction, we can state that decoding a codeword and decoding a syndrome for a generic linear code are NP-Hard. We note that, at the current state-of-the-art, no method faster than the search is known to solve the corresponding decision problem.

LEDAcrypt primitives rely on the indistinguishability of their generator matrix G and parity-check matrix H from the ones of a random linear code. At the moment of writing, the only technique known to exploit the non random nature of the H and G matrices of LEDAcrypt is to rely on the low-weight of the corresponding secret QC-LDPC to perform a guess on the structure on the underlying code.

2.3 Attacks based on exhaustive key search

Enumerating all the possible values for the secret key is, in principle, applicable to any cryptosystem. While the cost of performing an exhaustive key search is dominated by less computational demanding key recovery strategies in LEDAcrypt cryptosystems, we consider partial key enumeration attacks, aiming at scanning through all the possible H or Q sparse matrices as a support for other strategies. While there is no standing attack benefiting from an enumeration of possible H, from the enumeration of possible Q matrices or from the enumeration of both of them, we deem reasonable adding such a constraint to the design of the parameter sets as a peace-of-mind measure and obtain a system for which the said enumerations are computationally unfeasible.

We recall that H is a block circulant binary matrix constituted by $1 \times n_0$ circulant-blocks, each of which having size equal to p bits, while $n_0 \in \{2, 3, 4\}$ and p is a prime such that $\text{ord}_2(p) = p - 1$ (i.e., $2^{p-1} \bmod p = 1 \bmod p$). Q is a binary circulant-block matrix constituted by $n_0 \times n_0$ binary circulant-blocks with size p.

Considering that each row of a circulant-block of H has Hamming weight d_v, a straightforward counting argument yields $\sharp H = \binom{p}{d_v}^{n_0}$ as the number of possible choices for H. The number of possible choices for Q, denoted as $\sharp Q$, can be derived starting from the consideration that the weights of a row of each circulant block in a block-row of Q are equal for all the rows up to a rotation of the weights of the blocks. Such weights, denoted as $\{m_0, \ldots, m_{n_0-1}\}$, allow to write the number of possible choices for Q as $\sharp Q = \left[\prod_{i \in \{m_0, \ldots, m_{n_0-1}\}} \binom{p}{i} \right]^{n_0}$.

We also consider the possibility that an attacker performs an exhaustive key search employing a quantum computer. In such a case, the best scenario for the attacker is that it is possible to exploit Grover's algorithm to compute either H or Q, and test its correctness in deriving the other matrix and the corresponding public key. Assuming conservatively that the test can be implemented on

a quantum computer, we consider the resistance against exhaustive key search with a quantum computer to be the minimum between $\sqrt{\sharp H}$ and $\sqrt{\sharp Q}$ for the search over H and Q, respectively. In our approach, to prevent attacks relying on the partial exhaustive search for the value of either H or Q, we considered the remainder of the attack strategy which may be employed to derive the matrix which is not found via exhaustive search to have a constant complexity (i.e., $\Theta(1)$).

Therefore, we design the LEDAcrypt parameters such that any attack strategy which leverages the exhaustive search of H or Q to speedup a key recovery will, in turn, have a computational complexity matching or exceeding the required security level.

2.4 Attacks based on information set decoding

The most computationally effective technique known to attack the LEDAcrypt schemes is the same technique solving the Syndrome Decoding Problem (SDP) on random binary linear block codes and known as ISD. ISD was invented as a general efficient decoding technique for a random binary linear block code by Eugene Prange in [36]. While ISD is indeed more efficient than guessing the error affected positions in an incorrect codeword, its complexity is still exponential in the number of errors affecting the codeword.

ISD can thus be employed as a *message recovery attack* in both McEliece and Niederreiter cryptosystems: in the former, it will recover the error pattern from the error affected codeword constituting the ciphertext, allowing to recover the original message; while in the Niederreiter cryptosystem it will derive the error vector from a syndrome, under the assumption that it was added to a null codeword. When a message recovery attack of this kind is performed against a cryptosystem exploiting quasi cyclic codes, such as the case of LEDAcrypt, it is known that a speedup equal to the square root of the circulant block size can be achieved [39].

ISD algorithms have a long development history, dating back to the early '60s [36], and provide a way to recover the error pattern affecting a codeword of a generic random linear block code given a representation of the code in the form of either its generator or parity-check matrix.

Despite the fact that the improvement provided by ISD over the straightforward enumeration of all the possible error vectors affecting the codeword is only polynomial, employing an ISD technique provides substantial speedups. It is customary for ISD variant proposers to evaluate the effectiveness of their attacks considering the improvement on a worst-case scenario as far as the code rate and number of corrected errors goes (see, for instance [6]). Such an approach allows to derive the computational complexity as a function of a single variable, typically taken to be the code length n, and obtaining asymptotic bounds for the behavior of the algorithms.

In our parameter design, however, we chose to employ non-asymptotic estimates of the computational complexity of the ISD attacks. Therefore, we explicitly compute the amount of time employing a non-asymptotic analysis of the complexity of ISD algorithms, given the candidate parameters of the code at hand. This approach permits us to retain the freedom to pick rates for our codes which are different from the worst-case one for decoding, thus exploring different trade-offs in the choice of the system parameters. It is common for ISD variants to have free parameters, which should be tuned to achieve optimal performance. We sought the optimal case by explicitly computing the complexity of the ISD variant for a large region of the parameter space, where the minimum complexity resides. We consider the ISD variants proposed by Prange [36], Lee and Brickell [26],

Leon [27], Stern [41], Finiasz and Sendrier [12], and Becker, Joux, May and Meurer (BJMM) [6], in our computational complexity evaluation on classical computers. The reason for considering all of them is to avoid concerns on whether their computational complexity in the finite-length regime is already well approximated by their asymptotic behavior. For all the attacks, we consider the complexity as obtained with a logarithmic memory access cost, considering as the amount of required memory only the one taken by the lists in case of collision-based ISD techniques. Such a choice is mutuated by the fact that the size of such lists is exponential in the code parameters, and thus has a small but non negligible impact on the computation complexity.

In order to estimate the computational complexity of ISD on quantum computing machines, we consider the results reported in [9], which are the same employed in the original specification [4]. Since complete and detailed formulas are available only for the ISD algorithms proposed by Lee and Brickell, and Stern [41], we consider those as our computational complexity bound. While asymptotic bounds show that executing a quantum ISD derived from the May-Meurer-Thomae (MMT) algorithm [30] is faster than a quantum version of Stern's [22], we note that there is no computational complexity formulas showing this in the finite regime.

2.4.1 Key recovery attacks based on information set decoding

In the case of McEliece and Niederreiter cryptosystems instantiated with public codes characterized by sparse parity-check matrices, here of interest, ISD algorithms can also be used to mount a *key recovery attack*. In fact, in this case the dual of the public code contains low weight codewords that coincide with the rows of the sparse parity-check matrix of the public code. The latter can hence be recovered by searching for low codewords in the dual of the public code, and ISD can be exploited for this purpose. This indeed is a key recovery attack, since recovering a sparse parity-check matrix for the public code is sufficient for an attacker to perform efficient decoding.

In LEDAcrypt, the public key is the representation of a code $\mathcal{C}$ whose parity-check matrix is obtained as $L = HQ$, in the form

$$L = \left[L_0, L_1, \cdots, L_{n_0-1}\right], \tag{2.1}$$

where each block L_i is a $p \times p$ circulant with weight $d'_v \leq d_v m$.
It can be shown that $\mathcal{C}$ has minimum distance $2d'_v$; indeed, let us consider two distinct integers $0 \leq i_0, i_1 \leq n_0 - 1$, and define the $p \times n_0 p$ matrix C as follows

$$C = [C_0, \cdots, C_{n_0-1}], \text{ with } C_i \in \mathbb{F}_2^{p\times p}, \; C_i = \begin{cases} 0 & \text{if } i \neq i_0, i_1 \\ L_{i_1}^T & \text{if } i = i_0 \\ L_{i_0}^T & \text{if } i = i_1 \end{cases}. \tag{2.2}$$

It is easy to see that the rows of C are codewords of $\mathcal{C}$, as

$$CL^T = C_{i_0}L_{i_0}^T + C_{i_1}L_{i_1}^T = L_{i_1}^T L_{i_0}^T + L_{i_0}^T L_{i_1}^T = 0, \tag{2.3}$$

where the last equality is justified by the fact that multiplication between circulant matrices is commutative.

Let c be a row of a matrix C as in (2.2); if the attacker succeeds in determining c, he then succeeds in recovering two blocks of the secret matrix L. It is easy to see that this is enough to recover the whole L from the public key.

Each such codeword c has weight $2d'_v$, and can thus be searched by exploiting an ISD algorithm searching for low weight codewords.

In particular, in such a case the opponent needs to obtain the systematic generator matrix for $\mathcal{C}$, that is in the form

$$G_{\mathtt{sys}} = \left[\begin{array}{ccc|c} I_p & & & G_0 \\ & I_p & & G_1 \\ & & \ddots & \vdots \\ & & I_p & G_{n_0-2} \end{array}\right] = \left[\begin{array}{ccc|c} I_p & & & \left(L_0 L_{n_0-1}^{-1}\right)^T \\ & I_p & & \left(L_1 L_{n_0-1}^{-1}\right)^T \\ & & \ddots & \vdots \\ & & I_p & \left(L_{n_0-2} L_{n_0-1}^{-1}\right)^T \end{array}\right]. \tag{2.4}$$

Such a matrix corresponds to the public key of LEDAcrypt PKC, while it can be obtained with some simple computations from LEDAcrypt KEM. Then, a key recovery attack from $G_{\mathtt{sys}}$ can be performed in two different ways:

i. apply ISD on $G_{\mathtt{sys}}$, searching for codewords of weight $\leq 2d'_v$. In such a case, we have to consider that the ISD obtains a speed-up due to the fact that $\mathcal{C}$ contains multiple codewords of weight $2d'_v$, and the opponent is satisfied with each one of them. We have $\binom{n_0}{2}$ possible matrices C as in Eq. (2.2), each one containing p rows: thus, the number of codewords in $\mathcal{C}$ with weight $2d'_v$ (and the subsequent speed-up in ISD) is obtained as $p\binom{n_0}{2}$.

ii. choose a block G_i and form the matrix

$$G' = [I_p, G_i]. \tag{2.5}$$

The matrix G' corresponds to the generator matrix of a code $\mathcal{C}'$, with length $2p$ and dimension p, which contains codewords of weight $2d'_v$. Indeed, we have

$$L_{n_0-1}^T G' = [L_{n_0-1}^T; L_{n_0-1}^T \left(L_i L_{n_0-1}^T\right)^T] = [L_{n_0-1}^T; L_i^T]. \tag{2.6}$$

Thus, each information sequence equal to a row of $L_{n_0-1}^T$ corresponds to a codeword, in $\mathcal{C}'$, of weight $2d'_v$. Thus, an ISD algorithm can be applied on G', searching for such codewords.

Alternatively, the opponent might attack the dual code of $\mathcal{C}$, i.e, the code having L as one of its generator matrices. The rows of L have weight $d'_c = n_0 m d_v$, which is comparatively small with respect to the codeword length n. Any sparse row of L is a low-weight codeword belonging to the dual of the public code, and can thus be searched through an ISD algorithm.

Some potentially weak keys may exist, when the product between the matrices H and Q causes a too much large number of cancellations. In such a case, the matrix L will have a density that is slightly reduced than the maximum one: in other words, there will be circulant blocks L_i having row and column weight $< md_v$, and this will result in a row weight of L being $< n_0 m d_v$. The highest security level is achieved when the row weight of L is maximal so, in all these cases, ISD key - recovery attacks will be facilitated. Furthermore, having a fixed row weight for L facilitates constant-time implementation. In order to avoid these drawbacks, both LEDAcrypt KEM and LEDAcrypt PKC key generation phases employ a rejection sampling procedure, which is adopted to ensure that L always has maximal column weight (and, subsequently, maximum row weight). The key generation algorithm starts with the random extraction of matrices H and Q, which are subsequently multiplied to compute L; the rejection sampling is applied on L: if the obtained L

has L has row weight $< n_0 m d_v$, then it is discarded and a new pair is randomly selected. This way, we guarantee that ISD key recovery attacks have the highest achievable complexity. We point out that the complexity of the rejection sampling is particularly low and has, practically, a negligible impact on the system performances. In addition, the rejection rate of the keys is particularly low since, for the system parameters we have designed, randomly generated matrices H and Q lead to a full weight L with very high probability; a theoretical estimate of the rejection rate is provided in Appendix A.

We consider such key recovery attacks in our parameter design, evaluating their complexity for all the aforementioned ISD algorithms.

2.5 Attacks based on Bob's reactions

In addition to the proper sizing of the parameters of LEDAcrypt so that it withstands the aforementioned attacks, a last concern should be taken into account regarding the lifetime of a LEDAcrypt key pair, when keys are not ephemeral. In fact, whenever an attacker may gain access to a decryption oracle to which he may pose a large amount of queries, the so-called *reaction attack* becomes applicable. Reaction attacks recover the secret key by exploiting the inherent non-zero DFR of QC-LDPC codes [10, 11, 17]. In particular, these attacks exploit the correlation between the DFR of the code and the supports of the private matrices and the error vector used for encryption. Indeed, whenever e and H, Q or both of them have pairs of ones placed at the same distance, the decoder exhibits a DFR smaller than the average.

Such attacks require the collection of the outcome of decoding (success or failure) on a ciphertext for which the attacker knows the distances in the support of the error vector, for a significant number of ciphertexts, to achieve statistical confidence in the result. The information on the decoding status is commonly referred to as the *reaction* of the decoder, hence the name of the attack. In the following we briefly recall the work of [10]. The strongest countermeasure against these attacks is to choose a proper set of system parameters such that the DFR is negligible, which means an attacker would need an intolerably long time to observe a sufficient amount of decoding failures.

Given a binary vector v of length p, having $\Psi_v = \{p_0, p_1, \cdots, p_w\}$ as its support, i.e., the set of positions of v containing a symbol one, we define its distance spectrum $DS(v)$ as:

$$DS(v) = \{\min\{|p_i - p_j|, p - |p_i - p_j|\}, \quad p_i, p_j \in \Psi_v\} \tag{2.7}$$

For a circulant matrix C, the distance spectrum $DS(C)$ is defined as the distance spectrum of its first row, since all the rows share the same spectrum. Indeed, it can be easily shown that the cyclic shift of a vector does not change its distance spectrum. As proved in [10], it is possible to reconstruct a vector v once its distance spectrum and number of set symbols is known.

Reaction attacks against LEDAcrypt can exploit several strategies, as described in [37]. Let us focus on the j-th circulant block of L, i.e., $L_j = \sum_{i=0}^{n_0-1} H_i Q_{ij}$. If the attacker is able to recover $DS(L_j)$, for some $j \in [0; n_0 - 1]$, then L_j can be reconstructed and the whole matrix L recovered from the public key. Then, $L = HQ$ can be used to perform BF decoding over the public code. Alternatively, the attacker can aim at recovering $DS(H_i)$ and $DS(Q_{ij})$, $\forall i \in [0; n_0 - 1]$ and for some $j \in [0; n_0 - 1]$, in such a way as to reconstruct L_j from the knowledge of H and one column of Q.

In all these attacks, Eve generates a large number of valid plaintext/ciphertext pairs and sends the ciphertexts to Bob, asking for decryption. Then, the statistical analysis of Bob's reactions (in terms of decoding success or failure) is exploited by Eve to recover the distance spectra she is interested in. When an IND-CCA2 secure conversion is adopted, the error vector used during encryption cannot be chosen by Eve, and can be seen as a randomly extracted vector among all the possible $\binom{n}{t}$ n-tuples with weight t. A critical parameter for these attacks is T, which is the number of collected ciphertexts. In fact, after observing T ciphertexts, the average number of failures observed by Eve is $T \cdot \mathrm{DFR}$, which is the basis for her statistical analysis. For LEDAcrypt instances, we consider that any key pair has a lifetime equal to $T = \mathrm{DFR}^{-1}$, which means allowing that only one decryption failure is observed by Eve during the whole lifetime of each key pair, on average.

It has been recently pointed out in [35] that some mechanisms exist to generate error vectors able to artificially increase the DFR of these systems. However, they start from the observation of at least one failure, which is unlikely when the original DFR is sufficiently low. In addition, these methods require the manipulation of error vectors, which is not feasible when an IND-CCA2 secure conversion is adopted.

2.5.1 Effects on instances with ephemeral keys and accidental key reuse

LEDAcrypt KEM instances with Perfect Forward Secrecy (PFS) and IND-CPA exploit ephemeral keys that are renewed before each encryption. Hence, any key pair can be used to decrypt one ciphertext only. In such a case, statistical attacks based on Bob's reactions are inherently unfeasible on condition that the ephemeral nature of the keys is strictly preserved.

Reaction attacks could instead be attempted in the case of an accidental reuse of the keys in these instances. However, as shown in Section 3.1, the parameter choices of LEDAcrypt KEM instances with ephemeral keys guarantee a DFR in the order of 10^{-8}–10^{-9}. An attacker would need to collect DFR^{-1} ciphertexts encrypted with the same key, on average, before observing one decryption failure. Hence, these instances are protected against an accidental key reuse of significant length.

2.5.2 Effects on instances with long term keys

LEDAcrypt KEM and LEDAcrypt PKC instances with long term keys employ a suitable conversion to achieve IND-CCA2. The IND-CCA2 model assumes that an attacker is able to create an unlimited number of chosen ciphertexts and ask for their decryption to a decryption oracle owning the private key corresponding to the public key used for their generation. Under these assumptions, reaction attacks become possible, since Eve is able to observe a large number of decryptions related to the same key pair.

An effective approach to counteract reaction attacks is the use of a suitable conversion of the type described in Section 1.4.1, which yields a cryptosystem resistant against active attacks (IND-CCA2). When such a conversion is employed, the attacker is no longer free to choose the error vector used during encryption, which instead is a function of the encrypted message. Therefore, reaction attacks cannot be facilitated by choosing suitably forged error vectors that may be able to artificially increase the system DFR and accelerate the analysis of the distance spectrum of the secret key.

A noteworthy point is that the current existing IND-CCA2 constructions require the DFR of the scheme to be negligible. Indeed, most IND-CCA2 attaining constructions require the underlying cryptosystem to be correct, i.e., $D_{sk}(E_{pk}(m)) = m$, for all valid key pairs (pk, sk) and for all valid messages m. Recent works [20, 21] tackled the issue of proving a construction IND-CCA2 even in the case of an underlying cipher affected by decryption failures. The results obtained show that, in case the DFR is negligible in the security parameter, it is possible for the construction to attain IND-CCA2 guarantees even in case of decryption failures.

In systems with non-zero DFR, endowed with resistance against an Adaptive Chosen Ciphertext Attack (CCA2), attacks such as the modification of a ciphertext aimed at inducing the receiver's decoder in error are warded off. Therefore, our choice of employing an IND-CCA2 achieving construction to build both our PKC and KEM, paired with appropriate parameters guaranteeing a negligible DFR allows us to thwart ciphertext alteration attacks such as the ones pointed out in the official comments to our proposal during the first round[1].

[1] `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/LEDAkem-official-comment.pdf`

Chapter 3

LEDAcrypt Parameters

In this chapter we describe an automated procedure for the design of tight and optimal sets of parameters for the QC-LDPC codes employed in LEDAcrypt. This procedure is available as public domain software. Concerning the parametrization of the symmetric components involved in the LEDAcrypt primitives: we chose to employ as a PRNG NIST standard CTR-DRBG instantiated with AES-256 and $l_{\texttt{seed}}$ equal to 192, 256, and 320 bits for NIST Category 1,3,5 parameters, respectively. We chose to employ as a cryptographic hash NIST standard SHA-3 with digest length $l_{\texttt{hash}}$ 256, 384, and 512 bits for NIST category 1,3, and 5 parameters, respectively.

An open source software implementation of the routines for computing the complexity of the described attacks is available at `https://github.com/ledacrypt`. How this procedure can be used for designing LEDAcrypt instances with ephemeral and long term keys is shown in Sections 3.1 and 3.2, respectively.

The LEDAcrypt design procedure described in this section takes as input the desired security level λ_c and λ_q, expressed as the base-2 logarithm of the number of operations of the desired computational effort on a classical and quantum computer, respectively. In addition to λ_c and λ_q, the procedure also takes as input the number of circulant blocks, $n_0 \in \{2, 3, 4\}$, forming the parity-check matrix H, allowing tuning of the code rate. As a third and last parameter, the procedure takes as input the value of ϵ, which tunes the system DFR. We first consider instances with ephemeral keys, which are designed using $\epsilon = 0.3$: the resulting DFR values are in the range 10^{-9}–10^{-8}. Then, for instances using long term keys, the system parameters are updated as described in Section 3.2 in order to achieve a smaller DFR.

The parameter design procedure outputs the size of the circulant blocks, p, the weight of a column of H, d_v, the number of intentional errors, t, the weights of the n_0 blocks of a row of Q, i.e., $\langle m_0, m_1, \ldots, m_{n_0-1} \rangle$, with $\sum_{i=0}^{n_0-1} m_i = m$. The procedure enforces the following constraints on the parameter choice:

- Classical and quantum exhaustive searches for the values of H or Q should require at least 2^{λ_c} and 2^{λ_q} operations. This constraint binds the value of the circulant block p and the weight of a row of the circulant block, d_v for H and m_i for Q, to be large enough.
- The minimum cost for a message recovery via ISD on both quantum and classical computers must exceed 2^{λ_q} and 2^{λ_c} operations, respectively. This constraint binds the values of the code

length $n = n_0 p$, the code dimension $k = (n_0 - 1)p$ and the number of errors t to be chosen such that an ISD on the code $\mathcal{C}(n, k, t)$ requires more than 2^{λ_q} or 2^{λ_c} operations on a quantum and a classical computer.

- The minimum cost for a key recovery attack via ISD on both quantum and classical computers must exceed 2^{λ_q} and 2^{λ_c} operations, respectively. This constraint binds the values of the code length $n = n_0 p$, the code redundancy $r = p$ and the number of ones in a row of HQ, $d'_v n_0$, with $d'_v = d_v m$ to be chosen such that an ISD on the code $\mathcal{C}(n, r, d'_v n_0)$ requires more than 2^{λ_q} or 2^{λ_c} operations on a quantum and classical computer.
- The choice of the circulant block size, p, should be such that p is a prime number and such that $\mathrm{ord}_2(p) = p - 1$ (see Theorem 1.1.3).
- The choice of the circulant block size, p, and parity-check matrix density, $n_0 d_v$, must allow the code to correct the required amount of errors. This is tested through the computation of the decoding threshold, as described in the original specification [4].
- The weights of the circulant blocks of Q must guarantee the existence of its multiplicative inverse according to the criterion defined by Theorem 1.1.4, i.e., the permanent of the matrix of the block weights must be odd.

We report a synthetic description of the procedure implemented in the publicly available code as Algorithm 13. The rationale of the procedure[1] is to proceed in refining the choice for p, t, d_v, and all the m_i's at fix point, considering only values of p respecting $\mathrm{ord}_2(p) = p - 1$.

Since there are cyclic dependencies among the constraints on p, t, d_v and m, the search for the parameter set is structured as a fix point solver iterating on a test on the size of p (lines 2–28).

The loop starts by analyzing the next available prime p extracted from a list of pre-computed values such that $\mathrm{ord}_2(p) = p - 1$, and sorted in ascending order (line 3). The length, n, dimension, k, and redundancy, $r = n - k$, of the code are then assigned to obtain a code rate equal to $1 - \frac{1}{n_0}$ (line 4). Subsequently, the procedure for the parameter choice proceeds executing a loop (lines 5–7) to determine a value t, with $t < r$, such that a message recovery attack on a generic code $\mathcal{C}(n, k, t)$ requires more than the specified amount of computational efforts on both classical and quantum computers.

To determine the weight of a column of H, i.e., d_v and the weight of a column of Q, i.e., m, with $m = \sum_{i=0}^{n_0-1} m_i$, the procedure moves on searching for a candidate value of d'_v, where $d'_v = d_v m$ and $d'_v n_0$ is the weight of a row of HQ. Given a value for d'_v (line 8 and line 21), the value of d_v is computed as the smallest odd integer greater than the rounded value of the square root of d'_v (line 10). The condition of d_v being odd is sufficient to guarantee the non singularity of the circulant blocks of H, while the square root computation is meant to distribute the weight d'_v evenly between the weight of a column of H and the weight of a column of Q. The weight of a column of Q, i.e., m, is then computed through the loop in lines 11–15. Specifically, the value of m must allow a partition in n_0 integers (i.e., $m = \sum_{i=0}^{n_0-1} m_i$) such that the permanent of the circulant integer matrix having the said partition as a row is odd, for the matrix Q to be invertible [4]. Therefore, in the loop body the value of m is assumed as $\left\lceil \frac{d'_v}{d_v} \right\rceil$ (line 13) and subsequently checked to derive

[1] Note that, in the pseudocode of Algorithm 13, the loop construct **while**(< condition >) ... iterates the execution of instructions in the loop body when the condition is **true**, while the loop construct **Repeat** ... **until**(< condition >) iterates the instructions in the loop body when the condition is **false**.

Algorithm 13: LEDAcrypt Parameter Generation

Input: λ_c, λ_q:desired security levels against classical and quantum attacks, respectively;
ϵ: safety margin on the minimum size of a circulant block of the secret parity-check matrix H, named $p_{th} = p(1+\epsilon)$, where p is the size of a circulant block, so that the code is expected to correct all the errors with acceptable DFR;
n_0: number of circulant blocks of the $p \times n_0 p$ parity-check matrix H of the code. The Q matrix is constituted by $n_0 \times n_0$ circulant blocks as well, each of size p.

Output: p: size of a circulant block; t: number of errors; d_v: weight of a column of the parity matrix H;
$\langle m_0, m_1, \ldots, m_{n_0-1}\rangle$: an integer partition of m, the weight of a row of the matrix Q. Each m_i is the weight of a block of Q.

Data: NextPrime(x): subroutine returning the first prime p larger than the value of the input parameter and such that $\mathrm{ord}_2(p) = p-1$;
C-ISD-Cost(n,k,t), Q-ISD-Cost(n,k,t): subroutines returning the costs of the fastest ISDs employing a classical and a quantum computer, respectively;
$\sharp Q$: number of valid $n_0 p \times n_0 p$ block circulant matrices, $\sharp Q = \left(\prod_{i\in\{m_0,\ldots,m_{n_0-1}\}} \binom{p}{i}\right)^{n_0}$;
$\sharp H$: number of valid $p \times n_0 p$ block circulant matrices, $\sharp H = \binom{p}{d_v}^{n_0}$;
`FindmPartition`(m, n_0): subroutine returning two values. The former one is a sequence of numbers composed as the last integer partition of m in n_0 addends ordered according to the lexicographic order of the reverse sequences, i.e., $\langle m_0, m_1, \ldots, m_{n_0-1}\rangle$, (this allows to get a sequence of numbers as close as possible among them and sorted in decreasing order). The latter returned value is a Boolean value `PermanentOk` which points out if the partition is legit (**true**) or not (**false**).

1 $p \leftarrow 1$
2 **repeat**
3 $\quad p \leftarrow$ NextPrime(p)
4 $\quad n \leftarrow n_0 p$, $k \leftarrow (n_0-1)p$, $r \leftarrow p$
5 $\quad t \leftarrow 1$
6 $\quad$ **while** $\left(t \le r \wedge \left(\text{C-ISD-Cost}(n,k,t) < 2^{\lambda_c} \vee \text{Q-ISD-Cost}(n,k,t) < 2^{\lambda_q}\right)\right)$ **do**
7 $\quad\quad t \leftarrow t+1$
8 $\quad d'_v \leftarrow 4$
9 $\quad$ **repeat**
10 $\quad\quad d_v \leftarrow \left\lfloor \sqrt{d'_v} \right\rfloor - 1 - (\left\lfloor \sqrt{d'_v} \right\rfloor \bmod 2)$
11 $\quad\quad$ **repeat**
12 $\quad\quad\quad d_v \leftarrow d_v + 2$
13 $\quad\quad\quad m \leftarrow \left\lceil \frac{d'_v}{d_v} \right\rceil$
14 $\quad\quad\quad \langle m_0, m_1, \cdots, m_{n_0-1}\rangle, \texttt{PermanentOk} \leftarrow \texttt{FindmPartition}(m, n_0)$
15 $\quad\quad$ **until** `PermanentOk` = **true** $\vee$ $(m < n_0)$
16 $\quad\quad$ **if** $(m > n_0)$ **then**
17 $\quad\quad\quad \texttt{SecureOk} \leftarrow \text{C-ISD-Cost}(n, r, n_0 d'_v) \ge 2^{\lambda_c} \wedge \text{Q-ISD-Cost}(n, r, n_0 d'_v) \ge 2^{\lambda_q}$
18 $\quad\quad\quad \texttt{SecureOk} \leftarrow \texttt{SecureOk} \wedge \sharp H \ge 2^{\lambda_c} \wedge \sqrt{\sharp H} \ge 2^{\lambda_q} \wedge \sharp Q \ge 2^{\lambda_c} \wedge \sqrt{\sharp Q} \ge 2^{\lambda_q}$
19 $\quad\quad$ **else**
20 $\quad\quad\quad \texttt{SecureOk} \leftarrow$ **false**
21 $\quad\quad d'_v \leftarrow d'_v + 1$
22 $\quad$ **until** $(\texttt{SecureOk} = \textbf{true} \vee d'_v n_0 \ge p)$
23 $\quad$ **if** $(\texttt{SecureOk} = \textbf{true})$ **then**
24 $\quad\quad p_{th} \leftarrow \texttt{BF}_{\texttt{th}}(n_0, m d_v, t)$
25 $\quad$ **else**
26 $\quad\quad p_{th} \leftarrow p$
27 **until** $p > p_{th}(1+\epsilon)$
28 **return** $(p, t, d_v, m, \langle m_0, m_1, \cdots, m_{n_0-1}\rangle)$

the mentioned partition in n_0 integers. The loop (lines 11–15) ends when either a valid partition of m is found or m turns to be smaller than the number of blocks n_0 (as finding a partition in this case would be not possible increasing only the value of d_v).

Algorithm 13 proceeds to test for the security of the cryptosystem against key recovery attacks and key enumeration attacks on both classical and quantum computers (lines 16–18). If a legitimate value for m has not been found, the current parameters of the cryptoystem are deemed insecure

(line 20). In line 21, the current value of d'_v is incremented by one and another iteration of the loop is executed if the security constraints are not met with the current parameters (i.e., `SecureOk` = **false**) and it is still viable to perform another iteration to check the updated value of d'_v, i.e., $d'_v n_0 < p$ (line 22).

If suitable values for the code parameters from a security standpoint are found, the algorithm computes the minimum value of p, named p_{th}, such that the decoding algorithm is expected to correct t errors, according to the methodology reported in [4] (see lines 23–24); otherwise, the value of p_{th} is forced to be equal to p (lines 25–26) in such a way that another iteration of the outer loop of Algorithm 13 is executed through picking a larger value of p and new values for the remaining parameters.

We note that, while the decoding threshold provides a sensible estimate of the fact that the QC-LDPC code employing the generated parameters will correct the computed amount of errors, this is no substitute for a practical DFR evaluation, which is then performed through Montecarlo simulations. Willing to target a DFR of 10^{-9}, we enlarged heuristically the value of p until the target DFR was reached (in steps of 5% of the value found by the tool). The enlargement took place for:

- Category 1: $n_0 = 2$: 6 times, $n_0 = 3$: 1 time, $n_0 = 4$: 1 time
- Category 3: $n_0 = 2$: 4 times, $n_0 = 3$: 0 times, $n_0 = 4$: 0 times
- Category 5: $n_0 = 2$: 0 times, $n_0 = 3$: 0 times, $n_0 = 4$: 0 times.

The C++ tool provided follows the computation logic described in Algorithm 13, but it is optimized to reduce the computational effort as follows:

- The search for the values of t and d'_v respecting the constraints is performed by means of a dichotomic search instead of a linear scan of the range.
- The computations of the binomial coefficients employ a tunable memorized table to avoid repeated re-computation, plus a switch to Stirling's approximation (considering the approximation up to the fourth term of the series) only in the case where the value of $\binom{a}{b}$ is not available in the table and $b > 9$. In case the value of the binomial is not available in the table and $b < 9$ the result is computed with the iterative formula for the binomial, to avoid the discrepancies between Stirling's approximation and the actual value for small values of b.
- The values of p respecting the constraint $\mathrm{ord}_2(p) = p - 1$ are pre-computed up to $119,981$ and stored in a lookup table.
- The search for the value of p is not performed scanning linearly the aforementioned table. The strategy to find the desired p starts by setting the value of the candidate for the next iteration to NextPrime($\lceil (1+\epsilon)p_{th} \rceil$) up to finding a value of p, $\bar{p}$ which satisfies the constraints. Subsequently the algorithm starts scanning the list of primes linearly from $\bar{p}$ backwards to find the smallest prime which satisfies the constraints.

The C++ tool relies on Victor Shoup's NTL library (available at `https://www.shoup.net/ntl/`), in particular for the arbitrary precision integer computations and the tunable precision floating point computations, and requires a compiler supporting the C++11 standard.

Table 3.1: Parameter sizes for LEDAcrypt instances with ephemeral keys, obtained with the automated parameter design tool.

NIST Cat.	n_0	p	t	d_v	m	errors out of decodes
1	2	14,939	136	11	[4, 3]	14 out of $1.2 \cdot 10^9$
	3	7,853	86	9	[4, 3, 2]	0 out of $1 \cdot 10^9$
	4	7,547	69	13	[2, 2, 2, 1]	0 out of $1 \cdot 10^9$
3	2	25,693	199	13	[5, 3]	2 out of $1 \cdot 10^9$
	3	16,067	127	11	[4, 4, 3]	0 out of $1 \cdot 10^9$
	4	14,341	101	15	[3, 2, 2, 2]	0 out of $1 \cdot 10^9$
5	2	36,877	267	11	[7, 6]	0 out of $1 \cdot 10^9$
	3	27,437	169	15	[4, 4, 3]	0 out of $1 \cdot 10^9$
	4	22,691	134	13	[4, 3, 3, 3]	0 out of $1 \cdot 10^9$

Table 3.2: Computational cost of an exhaustive enumeration attack on either the matrix H or the matrix Q. The quantum execution model considers the possibility of attaining the full speedup yielded by the application of Grover's algorithm to the computation.

NIST Cat.	n_0	H Enumeration cost ($\log_2$ ♯binary op.s)		Q enumeration cost ($\log_2$ ♯quantum gates)	
		Classical	Quantum	Classical	Quantum
1	2	254.55	127.27	179.79	89.89
	3	295.93	147.96	326.84	163.42
	4	539.64	269.82	348.68	174.34
3	2	315.79	157.89	247.34	123.67
	3	385.30	192.65	425.80	212.90
	4	667.42	333.71	474.74	237.37
5	2	283.24	145.62	350.84	175.42
	3	542.70	271.35	451.27	225.63
	4	622.26	311.13	703.06	351.53

3.1 Parameters for LEDAcrypt instances with ephemeral keys

In Table 3.1 we provide parameters for LEDAcrypt KEM instances employing ephemeral keys. Deriving them took approximately a day for all the parameter sets with $n_0 \in \{3, 4\}$ and approximately a month for all the parameter sets with $n_0 = 2$ on a dual socket AMD EPYC 7551 32-Core CPU. The memory footprint for each parameter seeking process was below 100 MiB.

3.1.1 Resulting computational complexity of attacks

When an algorithmic procedure is exploited for the design of parameter sets, as in our case, some constraints on the choice of the row/column weights of H and Q must be imposed in such a way as to make enumeration of either H or Q unfeasible to an attacker. Therefore, enumeration attacks of the

Table 3.3: Cost of performing a message recovery attack, i.e., an ISD on the code $\mathcal{C}(n_0p, (n_0-1)p, t)$, for LEDAcrypt instances with the parameters p, t reported in Table 3.1, employing the considered ISD variants.

NIST Cat.	n_0	Classical computer ($\log_2$ ♯binary op.s)						Quantum computer ($\log_2$ ♯quantum gates)	
		Prange [36]	L-B [26]	Leon [27]	Stern [41]	F-S [12]	BJMM [6]	Q-LB [9]	Q-Stern [9]
1	2	169.05	158.23	156.35	148.59	148.57	144.37	97.26	98.67
	3	167.72	157.37	154.51	147.67	147.65	144.29	96.14	97.55
	4	169.62	159.40	155.86	149.32	149.31	145.98	97.47	98.22
3	2	234.11	222.19	220.26	210.42	210.41	207.17	130.22	131.62
	3	235.32	223.84	220.82	211.91	211.90	208.71	130.67	132.07
	4	235.98	224.66	220.97	212.39	212.39	209.10	131.26	132.66
5	2	303.56	290.79	288.84	277.40	277.39	274.54	165.18	166.58
	3	303.84	291.53	288.42	277.98	277.98	274.34	165.48	166.88
	4	303.68	291.54	287.78	277.67	277.67	274.91	165.52	166.92

type described in Section 2.3 must be taken into account. In Table 3.2 we report the computational cost of performing such an exhaustive enumeration, both with a classical and a quantum computer. The latter has been obtained by applying the speedup due to Grover's algorithm to the complexity computed considering a classical computer. From the results in Table 3.2 it is straightforward to note that an exhaustive search on either H or Q is clearly above the required computational effort.

Then, as described in Section 2.4.1, the two main attacks that can be mounted against the considered systems are message recovery attacks and key recovery attacks based on ISD algorithms. Table 3.3 and Table 3.4 report the complexity of these attacks against LEDAcrypt instances employing the parameters p, t in Table 3.1. An interesting point to be noted is that, while providing clear asymptotic speedups, the improvements to the ISD algorithms proposed since Stern's [41] are only able to achieve a speedup between 2^2 and 2^4 when their finite regime complexities are considered in the range of values concerning LEDAcrypt cryptosystem parameters. Concerning quantum ISD, it is interesting to notice that the quantum variant of the Stern's algorithm as described by de Vries [9] does not achieve an effective speedup when compared against a quantum transposition of Lee and Brickell's ISD. Such a result can be ascribed to the fact that the speedup obtained by the reduction in the number of ISD iterations which can be obtained by Stern's ISD is mitigated by the fact that the overall number of iterations to run is quadratically reduced by applying Grover's algorithm to execute them.

Comparing the computational complexities of the message recovery attack (Table 3.3) and the key recovery attack (Table 3.4), we note that performing a message recovery attack is almost always easier than the corresponding key recovery attack on the same parameter set, albeit by a small margin.

Table 3.4: Cost of performing a key recovery attack, i.e., an ISD on the code $\mathcal{C}(n_0 p, p, n_0 d_v m)$, for the revised values of the parameters p, n_0, d_v, m reported in Table 3.1, employing the considered ISD variants

NIST Cat.	n_0	Classical computer ($\log_2$ ♯binary op.s)						Quantum computer ($\log_2$ ♯quantum gates)	
		Prange [36]	L-B [26]	Leon [27]	Stern [41]	F-S [12]	BJMM [6]	Q-LB [9]	Q-Stern [9]
1	2	180.25	169.06	167.24	158.76	158.75	154.94	99.21	100.62
	3	169.36	157.78	156.53	147.71	147.68	144.08	93.93	95.34
	4	179.86	167.79	165.69	157.13	157.10	153.01	99.71	101.12
3	2	237.85	225.77	223.87	213.72	213.71	210.64	128.35	129.75
	3	241.70	228.98	227.03	216.59	216.57	213.18	130.56	131.96
	4	254.92	241.73	238.97	228.80	228.78	224.76	137.60	139.01
5	2	315.08	302.11	300.19	288.35	288.34	285.71	167.04	168.44
	3	320.55	306.93	304.48	292.78	292.77	289.00	170.31	171.71
	4	312.68	298.84	295.66	284.59	284.58	280.91	166.82	168.22

3.2 Parameters for LEDAcrypt instances with long term keys

For LEDAcrypt instances employing long term keys, we need that the DFR is sufficiently small to enable IND-CCA2. However, such small values of DFR cannot be assessed through Montecarlo simulations. Hence, for these instances we consider a Q-decoder performing two iterations and exploit the analysis reported in Section 1.6 in order to characterize its DFR.

To this end, we first employ an additional rejection sampling in the key generation phase, in order to ensure that the generated key pair can achieve, in the second decoder iteration, correction of all residual errors of weight $\leq \bar{t}$, where $\bar{t}$ is a properly chosen integer. Such a property can be obtained only after having tested the generated matrix L, since it must be such that the inequality $\mu(\bar{t}) + \mu(\bar{t} - 1) < md_v$ is satisfied (recall Equation (1.32)). For this purpose, during key generation each key pair is tested, by computing its characteristic values of $\mu(\bar{t})$ and $\mu(t - 1)$. If the above condition is not satisfied the generated key pair is discarded, and the procedure is repeated until a valid key pair is picked. For valid key pairs, achieving a desired target DFR value, $\overline{\text{DFR}}$, is guaranteed by the choice of code parameters such that the first iteration of the Q-decoder results in at most $\bar{t}$ residual errors with probability $> 1 - \overline{\text{DFR}}$.

Given a chosen target DFR and a set of parameters for a LEDAcrypt instance, we are able to evaluate the amount of secret keys which allow achieving the desired DFR target. Such a procedure is integrated in the key generation process for LEDAcrypt instances with long term keys, where the concern on the DFR is actually meaningful, as opposed to instances with ephemeral key pairs.

Some choices are reported in Table 3.5, by imposing a DFR smaller than 2^{-64} and $2^{-\lambda}$, where λ equals 128, 192, 256 for NIST Category 1, 3, 5, respectively. The proposed choices aim at parameter sets for which the probability of drawing a random secret key achieving the DFR target is significant. For designing these parameters, we start from the ones obtained through the automated parameter optimization procedure used for instances with ephemeral keys previously described, and keep the product md_v constant or slightly increased. Then, we proceed by increasing the size of the circulant blocks, until we obtain a probability smaller than the given target that the number of bit errors

that are left uncorrected by the first iteration is $\leq \bar{t}$. In particular, such a probability is computed by considering all possible choices for the flipping threshold of the first iteration, and by taking the optimal one (i.e., the one corresponding to the maximum value of the probability).

For any set of parameters so designed, we draw 100 key pairs at random, and evaluate how many of them satisfy the inequality $\mu(\bar{t}) + \mu(\bar{t}-1) < md_v$. As it can be seen from the results reported in Table 3.5, the parameter sets we determined are able to achieve a DFR $< 2^{-64}$ increasing the code size by a factor ranging from $2\times$ to $3\times$ with respect to the case of ephemeral key pairs.

The obtained LEDAcrypt parameterizations show that it is possible to achieve the desired DFR discarding an acceptable number of key pairs, given proper tuning of the parameters. The parameter derivation procedure for these LEDAcrypt instances can also be automated, which could be advantageous in terms of flexibility in finding optimal parameters for a given code size or key rejection rate.

Table 3.5: Parameters for the LEDAcrypt KEM-LT and the LEDAcrypt PKC employing a two-iteration Q-decoder matching a DFR equal to 2^{-64} and a DFR equal to $2^{-\lambda}$, where λ equals 128, 192, 256 for NIST Category 1, 3, 5, respectively

NIST Category	$\mathbf{n_0}$	**DFR**	**p**	**t**	$\mathbf{d_v}$	**m**	$\bar{\mathbf{t}}$	**No. of keys out of 100 providing the guaranteed DFR**	$\mathbf{b_0}$
1	2	2^{-64}	35,899	136	9	[5, 4]	4	95	44
	2	2^{-128}	52,147	136	9	[5, 4]	4	95	43
3	2	2^{-64}	57,899	199	11	[6, 5]	5	92	64
	2	2^{-192}	96,221	199	11	[6, 5]	5	92	64
5	2	2^{-64}	89,051	267	13	[7, 6]	6	93	89
	2	2^{-256}	152,267	267	13	[7, 6]	6	93	88

Chapter 4

Performance of the LEDAcrypt primitives

4.1 Performance of the LEDAcrypt primitives

Willing to provide a preliminary gauge of the performance and key size improvements obtained from the new parameter sets proposed in this document, we report the results of the execution of both LEDAcrypt KEM and LEDAcrypt PKC employing the parameter sets presented in Table 3.1. The results provided in the following are obtained with a codebase exploiting the availability of *compiler intrinsics* to employ the AVX2 extended instruction set provided by x86_64 CPUs. The implementations provided in the `Optimized_Implementation` folder automatically enable the intrinsics-based code in case the presence of an AVX2 endowed CPU is suggested by the `gcc` compiler via automatically defined macros. The optimized codebase falls back to a C implementation for the sake of compatibility in case the AVX2 instruction set extensions are not available.

Table 4.1 reports the running times of the ephemeral key exchange primitive. The execution time of the ephemeraly key exchange is dominated by the time taken to perform the multiplicative inverse during key generation time and the decoder computation at decryption time. The encryption primitive is significantly fast, allowing around 25k encryptions per second for NIST Category 1 parameters on our platform. The total execution time of a KEM is between 1 and 2 ms, for NIST Category 1, which is a fraction of the latency of the typical network communication.

Table 4.2 reports the size of the key pairs and encapsulated secrets with our ephemeral KEM. We note that the minimum total transmitted size is achieved for NIST Category 1 and $n_0 = 3$ with 3120B, while the smallest encapsulated secret is slightly less than 1 kB.

Table 4.3 reports the running time of the instances of LEDAcrypt KEM with parameters providing negligible DFR. In particular we selected a 2^{-64} level to match the number of queries allowed to an attacker, and a DFR such that obtaining a decoding failure requires the same amount of effort of randomly guessing the secret key. The encapsulation and decapsulation timings are significantly smaller than the key generation due to the impact of the rejection sampling procedure validating the DFR for the selected key pair. We note that the current implementation of the said rejection sampling strategy is not exploiting the circulant properties of the Γ matrix on which it acts, and therefore is amenable to significant speedups. Indeed, the remaining portion of the key generation

Table 4.1: LEDAcrypt KEM with ephemeral keys – Running times for key generation, key encapsulation and key decapsulation, followed by the total time needed for a key exchange without considering transmission times as a function of the NIST category and the number of circulant blocks n_0 on an Intel Skylake i5-6600 at 3.6 GHz.
The figures are taken employing the completely portable reference implementation in ISO C11, compiled with `GCC` 6.3.0, employing `-march=native -O3` as optimization parameters

NIST Category	n_0	KeyGen (ms)	Encap. (ms)	Decap. (ms)	Total exec. time (ms)
1	2	1.374 (± 0.130)	0.046 (± 0.009)	0.340 (± 0.072)	1.759
	3	0.569 (± 0.089)	0.038 (± 0.012)	0.424 (± 0.057)	1.031
	4	0.884 (± 0.510)	0.043 (± 0.006)	1.305 (± 0.136)	2.233
3	2	3.725 (± 0.188)	0.092 (± 0.018)	0.950 (± 0.103)	4.768
	3	1.793 (± 0.271)	0.088 (± 0.023)	1.112 (± 0.075)	2.992
	4	2.759 (± 1.170)	0.112 (± 0.014)	2.065 (± 0.176)	4.936
5	2	7.644 (± 0.261)	0.176 (± 0.022)	1.276 (± 0.105)	9.095
	3	4.964 (± 0.602)	0.177 (± 0.013)	1.623 (± 0.122)	6.764
	4	5.649 (± 1.846)	0.217 (± 0.018)	2.752 (± 0.176)	8.618

Table 4.2: LEDAcrypt KEM with ephemeral keys – Sizes in bytes of the key pair (at rest and in memory), of the encapsulated secret and of the shared secret, as a function of the NIST category and the number of circulant blocks n_0

NIST Category	n_0	Private key (B)		Public key (B)	Encapsulated secret size (B)	Shared secret size (B)
		At rest	In memory			
1	2	24	452	1,872	1,872	32
	3	24	540	2,080	1,040	32
	4	24	684	2,832	944	32
3	2	32	644	3,216	3,216	48
	3	32	748	4,032	2,016	48
	4	32	924	5,400	1,800	48
5	2	40	764	4,616	4,616	64
	3	40	972	6,864	3,432	64
	4	40	1,092	8,520	2,840	64

procedure accounts for less than 1% of its total time. The encapsulation and decapsulation timings can be further reduced changing the polynomial multiplication strategy from the current Toom-Cook with 3 limbs to a Toom-Cook with 4 limbs, and improving the vectorization of the decoder, respectively. Table 4.4 reports the size of the key pairs of LEDAcrypt KEM-LT, showing that, for NIST Category 1, the public key size does not exceed 4.5 kB, thus representing a reasonable overhead in most message transmissions. The size of the private key at rest takes into account the storage of the secret DRBG seed plus a 8 bit wide counter of the number of rejections which took place during key generation, so that the corresponding tests can be skipped at decryption time, when the private key is regenerated starting from the DRBG seed.

Finally, Tables 4.5 and 4.6 report the running times and keysizes of LEDAcrypt PKC. A noteworthy point is that the minimum ciphertext overhead is around the size of the public key of the scheme

itself. Such a feature is achieved thanks to the Kobara-Imai-γ construction which allows to encrypt a portion of the plaintext within the asymmetric cipher payload, instead of encapsulating just a session key.

Table 4.3: LEDAcrypt KEM-LT – Running times for key generation, encap. and decaps.
Execution times on an Intel Skylake i5-6600 at 3.6 GHz are reported as a function of the NIST category and of the decryption failure rate provided by the choice of the parameters of the underlying QC-LDPC code.
The figures are taken employing the completely portable reference implementation in ISO C11, compiled with `GCC` 6.3.0, employing `-march=native -O3` as optimization parameters

NIST Category	n_0	DFR	KeyGen (s)	Encapsulation (ms)	Decapsulation (ms)
1	2	2^{-64}	0.295 (± 0.003)	0.132 (± 0.004)	0.417 (± 0.048)
	2	2^{-128}	0.549 (± 0.228)	0.163 (± 0.004)	0.549 (± 0.029)
3	2	2^{-64}	0.906 (± 0.007)	0.259 (± 0.005)	0.911 (± 0.180)
	2	2^{-192}	1.532 (± 0.082)	0.540 (± 0.007)	1.248 (± 0.155)
5	2	2^{-64}	2.521 (± 0.069)	0.685 (± 0.144)	1.411 (± 0.043)
	2	2^{-256}	4.252 (± 0.114)	0.845 (± 0.022)	2.284 (± 0.194)

Table 4.4: LEDAcrypt KEM-LT – Sizes in bytes of the key pair, of the encapsulated secret and of the shared secret, as a function of the NIST category and of the decryption failure rate provided by the choice of the parameters of the underlying QC-LDPC code.

NIST Category	n_0	DFR	Private key (B)		Public key (B)	Encapsulated secret size (B)	Shared secret size (B)
			At rest	In memory			
1	2	2^{-64}	25	468	4,488	4,488	32
	2	2^{-128}	25	468	6,520	6,520	32
3	2	2^{-64}	33	660	7,240	7,240	48
	2	2^{-192}	33	660	12,032	12,032	48
5	2	2^{-64}	41	884	11,136	11,136	64
	2	2^{-256}	41	884	19,040	19,040	64

Table 4.5: LEDAcrypt PKC – Running times for key generation, encryption and decryption assuming a plaintext message to be encrypted with size 1KiB.
Execution times on an Intel Skylake i5-6600 at 3.6 GHz are reported as a function of the NIST category and of the decryption failure rate provided by the choice of the parameters of the underlying QC-LDPC code.
The figures are taken employing the completely portable reference implementation in ISO C11, compiled with `GCC` 6.3.0, employing `-march=native -O3` as optimization parameters

NIST Category	n_0	DFR	KeyGen (s)	Encryption (ms)	Decryption (ms)
1	2	2^{-64}	0.290 (± 0.008)	0.29 (± 0.00)	0.76 (± 0.00)
	2	2^{-128}	0.422 (± 0.014)	0.42 (± 0.03)	1.18 (± 0.12)
3	2	2^{-64}	1.187 (± 0.483)	0.56 (± 0.11)	1.70 (± 0.21)
	2	2^{-192}	1.538 (± 0.043)	1.10 (± 0.11)	2.39 (± 0.07)
5	2	2^{-64}	2.543 (± 0.037)	1.02 (± 0.09)	3.26 (± 0.44)
	2	2^{-256}	4.240 (± 0.069)	1.53 (± 0.07)	4.16 (± 0.09)

Table 4.6: LEDAcrypt PKC – Sizes in bytes of the key pair, and the minimum and maximum ciphertext expansion overhead, as a function of the NIST category and of the decryption failure rate provided by the choice of the parameters of the underlying QC-LDPC code

NIST Category	n_0	DFR	Private key (B)		Public key (B)	Min. ciphertext overhead (B)	Max. ciphertext overhead (B)
			At rest	In memory			
1	2	2^{-64}	25	468	4,488	4,521	8,976
	2	2^{-128}	25	468	6,520	6,554	13,040
3	2	2^{-64}	33	660	7,240	7,283	14,480
	2	2^{-192}	33	660	12,032	12,077	24,064
5	2	2^{-64}	41	884	11,136	11,189	22,272
	2	2^{-256}	41	884	19,040	19,095	38,080

4.2 Known Answer Test values

Known answer tests generated for 100 runs of LEDAcrypt KEM with ephemeral keys can be found in the `KAT` directory of the submission package. The naming convention of the `req/rsp` file pairs is the following:

`PQCkemKAT_<private_key_size>_<value_of_the_n0_parameter>.req`
`PQCkemKAT_<private_key_size>_<value_of_the_n0_parameter>.rsp`

Known answer tests generated for 10 runs of LEDAcrypt KEM with long term keys can be found in the `KAT` directory of the submission package. The naming convention of the `req/rsp` file pairs is the following:

`PQCkem-LT-KAT_<private_key_size>_<DFR_equal_to_SL>.req`
`PQCkem-LT-KAT_<private_key_size>_<DFR_equal_to_SL>.rsp`

where DFR_equal_to_SL is set to 0 for the parameter sets with a DFR of 2^{-64}, and to 1 for the other parameter sets.

Known answer tests generated for 10 runs of LEDAcrypt PKC can be found in the `KAT` directory of the submission package. The naming convention of the `req/rsp` file pairs is the following:

`PQCencryptKAT_<private_key_size>_<DFR_equal_to_SL>.req`
`PQCencryptKAT_<private_key_size>_<DFR_equal_to_SL>.rsp`

where DFR_equal_to_SL is set to 0 for the parameter sets with a DFR of 2^{-64}, and to 1 for the other parameter sets.

Chapter 5

Summary of advantages and limitations

+ Built on an NP-complete problem under reasonable assumptions.
− Exposes a public Quasi-Cyclic Moderate-Density Parity-Check (QC-MDPC) code obtained as the combination of a secret QC-LDPC code with a secret transformation matrix (Q), and this yields additional structure with respect to randomly generated QC-MDPC codes.
+ Rejection sampling is adopted to guarantee that the said additional structure does not introduce any polarization in the generation of keys.
+ The said additional structure allows using new decoders (Q-decoders) that are significantly faster than classic bit flipping decoders.
+ Compact key pairs (below 23 kiB at most), minimum size private keys.
+ Decoders with further improved and theoretically foreseeable performance could be developed.
+ Requires only addition and multiplication over $\mathbb{F}_2[x]$ (modular inverse over $\mathbb{F}_2[x]/\langle x^p+1\rangle$ can be avoided), besides single-precision integer operations.
+ Fully patent free, self contained, public domain codebase written in ANSI-C11.
+ Easy to integrate in existing cryptographic libraries.
+ Particularly efficient to apply countermeasures against non-profiled power consumption and electromagnetic emissions side channel attacks.

Bibliography

[1] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, and Y.-K. Liu, "Status report on the first round of the NIST post-quantum cryptography standardization process," National Institute of Standards and Technology, Tech. Rep. NISTIR 8240, Jan. 2019.

[2] M. Baldi and F. Chiaraluce, "Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC codes," in *Proc. IEEE International Symposium on Information Theory (ISIT 2007)*, Nice, France, Jun. 2007, pp. 2591–2595.

[3] M. Baldi, *QC-LDPC Code-Based Cryptography*, ser. SpringerBriefs in Electrical and Computer Engineering. Springer International Publishing, 2014.

[4] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "LEDAkem and LEDApkc website," https://www.ledacrypt.org/.

[5] M. Baldi, M. Bianchi, and F. Chiaraluce, "Security and complexity of the McEliece cryptosystem based on QC-LDPC codes," *IET Information Security*, vol. 7, no. 3, pp. 212–220, Sep. 2012.

[6] A. Becker, A. Joux, A. May, and A. Meurer, "Decoding random binary linear codes in $2^n/20$: How $1 + 1 = 0$ improves information set decoding," in *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, 2012, pp. 520–536.

[7] E. Berlekamp, R. McEliece, and H. van Tilborg, "On the inherent intractability of certain coding problems," *IEEE Trans. Information Theory*, vol. 24, no. 3, pp. 384–386, May 1978.

[8] E. Bernstein and U. V. Vazirani, "Quantum complexity theory," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1411–1473, Oct. 1997.

[9] S. de Vries, "Achieving 128-bit security against quantum attacks in OpenVPN," Master's thesis, University of Twente, Aug. 2016. [Online]. Available: http://essay.utwente.nl/70677/

[10] T. Fabšič, V. Hromada, P. Stankovski, P. Zajac, Q. Guo, and T. Johansson, "A reaction attack on the QC-LDPC McEliece cryptosystem," in *Post-Quantum Cryptography: 8th International Workshop, PQCrypto 2017*, T. Lange and T. Takagi, Eds. Utrecht, The Netherlands: Springer International Publishing, Jun. 2017, pp. 51–68.

[11] T. Fabsic, V. Hromada, and P. Zajac, "A reaction attack on LEDApkc," Cryptology ePrint Archive, Report 2018/140, 2018, https://eprint.iacr.org/2018/140.

[12] M. Finiasz and N. Sendrier, "Security bounds for the design of code-based cryptosystems," in *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, 2009, pp. 88–105.

[13] R. G. Gallager, *Low-Density Parity-Check Codes*. M.I.T. Press, 1963.

[14] S. W. Golomb, "Run-length encodings," *IEEE Trans. Information Theory*, vol. 12, no. 3, pp. 399–401, 1966. [Online]. Available: https://doi.org/10.1109/TIT.1966.1053907

[15] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, "Applying Grover's Algorithm to AES: Quantum Resource Estimates," in *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, 2016, pp. 29–43.

[16] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annual ACM Symposium on the Theory of Computing*, Philadephia, PA, May 1996, pp. 212–219.

[17] Q. Guo, T. Johansson, and P. Stankovski Wagner, "A key recovery reaction attack on QC-MDPC," *IEEE Trans. Information Theory*, vol. 65, no. 3, pp. 1845–1861, Mar. 2019.

[18] Q. Guo, T. Johansson, and P. Stankovski, "A key recovery attack on MDPC with CCA security using decoding errors," in *Advances in Cryptology – ASIACRYPT 2016*, ser. Lecture Notes in Computer Science, J. H. Cheon and T. Takagi, Eds. Springer Berlin Heidelberg, 2016, vol. 10031, pp. 789–815.

[19] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the Fujisaki-Okamoto transformation," in *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, Y. Kalai and L. Reyzin, Eds., vol. 10677. Springer, 2017, pp. 341–371. [Online]. Available: https://doi.org/10.1007/978-3-319-70500-2_12

[20] H. Jiang, Z. Zhang, L. Chen, H. Wang, and Z. Ma, "IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10993. Springer, 2018, pp. 96–125. [Online]. Available: https://doi.org/10.1007/978-3-319-96878-0_4

[21] H. Jiang, Z. Zhang, and Z. Ma, "Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model," Cryptology ePrint Archive, Report 2019/134, to appear in PQCrypto 2019, 2019, https://eprint.iacr.org/2019/134.

[22] G. Kachigar and J. Tillich, "Quantum information set decoding algorithms," in *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, 2017, pp. 69–89.

[23] D. E. Knuth, *The Art of Computer Programming: Generating All Combinations and Partitions*, 1st ed. Addison-Wesley, 2005, vol. 4.

[24] K. Kobara, "Code-based public-key cryptosystems and their applications," in *Information Theoretic Security*, ser. Lecture Notes in Computer Science. Springer Verlag, 2010, vol. 5973, pp. 45–55.

[25] K. Kobara and H. Imai, "Semantically secure McEliece public-key cryptosystems — conversions for McEliece PKC," *Lecture Notes in Computer Science*, vol. 1992, pp. 19–35, 2001. [Online]. Available: citeseer.ist.psu.edu/kobara01semantically.html

[26] P. J. Lee and E. F. Brickell, "An observation on the security of McEliece's public-key cryptosystem," in *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, 1988, pp. 275–280.

[27] J. S. Leon, "A probabilistic algorithm for computing minimum weights of large error-correcting codes," *IEEE Trans. Information Theory*, vol. 34, no. 5, pp. 1354–1359, 1988.

[28] Y. X. Li, R. Deng, and X. M. Wang, "On the equivalence of McEliece's and Niederreiter's public-key cryptosystems," *IEEE Trans. Information Theory*, vol. 40, no. 1, pp. 271–273, Jan. 1994.

[29] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[30] A. May, A. Meurer, and E. Thomae, "Decoding random linear codes in $\tilde{O}(2^{0.054n})$," in *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, 2011, pp. 107–124.

[31] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory." *DSN Progress Report*, pp. 114–116, 1978.

[32] National Institute of Standards and Technology. (2016, Dec.) Post-quantum crypto project. [Online]. Available: http://csrc.nist.gov/groups/ST/post-quantum-crypto/

[33] ——. (2018) Post-quantum cryptography - round 1 submissions. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions

[34] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *Probl. Contr. and Inform. Theory*, vol. 15, pp. 159–166, 1986.

[35] A. Nilsson, T. Johansson, and P. Stankovski Wagner, "Error amplification in code-based cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 238–258, nov 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7340

[36] E. Prange, "The use of information sets in decoding cyclic codes," *IRE Trans. Information Theory*, vol. 8, no. 5, pp. 5–9, 1962.

[37] P. Santini, M. Baldi, and F. Chiaraluce, "Assessing and countering reaction attacks against post-quantum public-key cryptosystems based on QC-LDPC codes," in *Cryptology and Network Security*, ser. Lecture Notes in Computer Science, J. Camenisch and P. Papadimitratos, Eds., vol. 11124. Cham: Springer International Publishing, 2018, pp. 323–343.

[38] N. Sendrier, "Encoding information into constant weight words," in *Proceedings. International Symposium on Information Theory, 2005 (ISIT 2005)*, Sep. 2005, pp. 435–438.

[39] ——, "Decoding one out of many," in *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, 2011, pp. 51–67.

[40] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997.

[41] J. Stern, "A method for finding codewords of small weight," in *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, 1988, pp. 106–113.

[42] R. Townsend and E. J. Weldon, "Self-orthogonal quasi-cyclic codes," *IEEE Trans. Information Theory*, vol. 13, no. 2, pp. 183–195, Apr. 1967.

[43] R. Ueno, S. Morioka, N. Homma, and T. Aoki, "A high throughput/gate AES hardware architecture by compressing encryption and decryption datapaths - Toward efficient CBC-mode implementation," in *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, 2016, pp. 538–558.

[44] C. Xing and S. Ling, *Coding Theory: A First Course*. New York, NY, USA: Cambridge University Press, 2003.

Appendix A

Estimate of the rejection rate

We here provide a theoretical estimate of the number of keys that get discarded due to rejection sampling that is performed during key generation in order to ensure that the obtained matrix L has full weight.

First of all, we define $Pr_{\oplus}(p, v_1, v_2, v_x)$ as the probability that the sum of two random length-p vectors with weights v_1 and v_2 results in a vector with weight v_x; we have

$$Pr_{\oplus}(p, v_1, v_2, v_x) = \begin{cases} \frac{\binom{v_1}{\frac{v_1+v_2-v_x}{2}}\binom{p-v_1}{v_2-\frac{v_1+v_2-v_x}{2}}}{\binom{p}{v_2}}, & \text{if } \begin{cases} \max\{v_1, v_2\} - \min\{v_1, v_2\} \leq v_x \leq v_1 + v_2, \\ v_x \mod 2 = v_1 + v_2 \mod 2, \end{cases} \\ 0, & \text{otherwise.} \end{cases} \tag{A.1}$$

Let $Pr_{\oplus}^{(N)}(p, v, v_x)$ be the probability that the sum of N random length-p vectors with weight v results in a vector with weight v_x. This probability can be recursively defined as

$$P_{\oplus}^{(N)}(p, v, v_x) \begin{cases} \begin{cases} 0 & \text{if } v^{(0)} \neq v, \\ 1 & \text{if } v^{(0)} = v, \end{cases} & \text{for } N = 1, \\ \sum_{v^{(N-1)}=0}^{p} P_{\oplus}^{(N-1)}(p, v, v^{(N-1)}) P_{\oplus}(p, v, v^{(N-1)}, v_x), & \text{for } N \geq 2. \end{cases} \tag{A.2}$$

The above probabilities can be used to estimate the row weight distribution of L. First of all, we remind that each circulant block L_i in L, for $i = 0, 1, \cdots, n_0 - 1$, is obtained as $L_i = \sum_{j=0}^{n_0-1} H_j Q_{j,i}$. We also remind that all the blocks H_i have row and column weight d_v, while each block $Q_{j,i}$ has row and column weight $m_{j,i}$. Each product $H_i Q_{j,i}$ can be described as the sum of $m_{j,i}$ random circulant blocks with row weight d_v, and thus its weight distribution can be estimated through Eq. (A.2), with $N = m_{j,i}$ and $v = d_v$. The weight distribution of L_i can then be computed as $P_i^{(n_0-1)}(v_x)$, where the function P_i is defined through the following recursive expression

$$P_i^{(j)}(v_x) = \begin{cases} P_{\oplus}^{(m_{0,i})}(p, d_v, v_x), & \text{if } j = 0, \\ \sum_{v_x=0}^{p} \sum_{v_1=0}^{p} \sum_{v_2=0}^{p} P_i^{(j-1)}(v_1) P_{\oplus}^{(m_{j,i})}(p, d_v, v_2) P_{\oplus}(p, v_1, v_2, v_x), & \text{otherwise.} \end{cases} \tag{A.3}$$

We finally obtain the weight distribution of a row of L as

$$P_L(w_x) = \prod_{i=0}^{n_0-1} P_i^{(n_0-1)}(v_i), \quad \forall v_0, \cdots, v_{n_0-1} \text{ s.t. } \sum_{i=0}^{n_0-1} v_i = w_x. \tag{A.4}$$

The rejection rate is hence estimated as

$$\eta = 1 - P_L(n_0 m d_v). \tag{A.5}$$

Statements

Statement by Each Submitter

I, Marco Baldi, of Università Politecnica delle Marche, Dipartimento di Ingegneria dell'Informazione (DII), Via Brecce Bianche 12, I-60131, Ancona, Italy, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt, is my own original work, or if submitted jointly with others, is the original work of the joint submitters. I further declare that I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt.

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment.

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3, below, for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed:

Title:

Date:

Place:

Statement by Reference/Optimized Implementations' Owner(s)

I, Marco Baldi, of Università Politecnica delle Marche, Dipartimento di Ingegneria dell'Informazione (DII), Via Brecce Bianche 12, I-60131, Ancona, Italy, am one of the owners of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the post-quantum algorithm public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed:

Title:

Date:

Place:

Statement by Each Submitter

I, Alessandro Barenghi, of Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Via G. Ponzio 34/5, I-20133, Milano, Italy, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt, is my own original work, or if submitted jointly with others, is the original work of the joint submitters. I further declare that I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt.

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment.

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3, below, for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed:

Title:

Date:

Place:

Statement by Reference/Optimized Implementations' Owner(s)

I, Alessandro Barenghi, of Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Via G. Ponzio 34/5, I-20133, Milano, Italy, am one of the owners of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the post-quantum algorithm public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed:

Title:

Date:

Place:

Statement by Each Submitter

I, Franco Chiaraluce, of Università Politecnica delle Marche, Dipartimento di Ingegneria dell'Informazione (DII), Via Brecce Bianche 12, I-60131, Ancona, Italy, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt, is my own original work, or if submitted jointly with others, is the original work of the joint submitters. I further declare that I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt.

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment.

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3, below, for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed:

Title:

Date:

Place:

Statement by Reference/Optimized Implementations' Owner(s)

I, Franco Chiaraluce, of Università Politecnica delle Marche, Dipartimento di Ingegneria dell'Informazione (DII), Via Brecce Bianche 12, I-60131, Ancona, Italy, am one of the owners of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the post-quantum algorithm public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed:

Title:

Date:

Place:

Statement by Each Submitter

I, Gerardo Pelosi, of Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Via G. Ponzio 34/5, I-20133, Milano, Italy, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt, is my own original work, or if submitted jointly with others, is the original work of the joint submitters. I further declare that I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt.

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment.

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3, below, for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed:

Title:

Date:

Place:

Statement by Reference/Optimized Implementations' Owner(s)

I, Gerardo Pelosi, of Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Via G. Ponzio 34/5, I-20133, Milano, Italy, am one of the owners of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the post-quantum algorithm public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed:

Title:

Date:

Place:

Statement by Each Submitter

I, Paolo Santini, of Università Politecnica delle Marche, Dipartimento di Ingegneria dell'Informazione (DII), Via Brecce Bianche 12, I-60131, Ancona, Italy, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt, is my own original work, or if submitted jointly with others, is the original work of the joint submitters. I further declare that I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LEDAcrypt.

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment.

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3, below, for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed:

Title:

Date:

Place:

Statement by Reference/Optimized Implementations' Owner(s)

I, Paolo Santini, of Università Politecnica delle Marche, Dipartimento di Ingegneria dell'Informazione (DII), Via Brecce Bianche 12, I-60131, Ancona, Italy, am one of the owners of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the post-quantum algorithm public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed:

Title:

Date:

Place: