Name of the proposed cryptosystem

LEDApkc (Low-dEnsity parity-check coDe-bAsed public-key cryptosystem)

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 submitter.

Name of the owner, if any, of the cryptosystem

Same as submitter.

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

×

See also printed version of "Statement by Each Submitter".

LEDApkc: Low-dEnsity parity-check coDe-bAsed public-key cryptosystem

Specification revision 1.0 – November 30, 2017

Contents

Chapter 1

Complete written specification

LEDApkc is a public-key cryptosystem (PKC) built from the McEliece cryptosystem based on linear error-correcting codes. In particular, LEDApkc exploits the advantages of relying on quasi-cyclic low-density parity-check (QC-LDPC) codes providing high decoding speeds and compact keypairs [4, 5], with three main innovations:

i. Reaction attacks of the type presented in [15] are taken into account and a secure lifetime for a keypair is estimated.

ii. A new decoding algorithm is designed: it provides faster decoding than the regular bit flipping (BF) decoding procedure, saves a computationally demanding matrix inverse computation, and allows a reduction in the required private key storage.

iii. A fully fledged conversion of the type described in [22, 23] is implemented to achieve indistinguishability under adaptive chosen ciphertext attack (IND-CCA2).

The main known attacks against this system are those applicable against QC-LDPC code-based cryptosystems [4], which have been studied for ten years since the first proposal appeared in [3], plus statistical attacks recently introduced in [15, 19]. We carefully analyze their capabilities and provide parametrization for the LEDApkc system 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

We now provide a set of background notions and nomenclature concerning binary error correcting codes, and in particular Low-Density Parity-Check codes, which are the foundational constructs of LEDApkc.

1.1.1 Linear error correcting codes

Binary error correcting codes rely on a redundant representation of information in the form of binary string to be able to detect and correct accidental bit errors which may happen during transmission or storage. We will 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 from now on simply as binary codes.
In this setting, let $\mathbb{F}_2$ be the binary finite field with the addition and multiplication operations which corresponds 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 $c \in \mathbb{F}_2^k$. The decoding process, instead, given a codeword $\hat{c}$ corrupted by an error vector $e \in \mathbb{F}_2^k$ with Hamming weight $t > 0$ ($\hat{c} = c + e$), 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 $\tilde{c}$ there is a decoding procedure to retrieve both the error vector e and the original information word u.

Definition 1.1.1 (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 1.1.1 is that the sum modulo 2, i.e., the component wise exclusive-or, of two codewords is also a codeword.

Definition 1.1.2 (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. 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

$$c = u_0 g_0 + \mathrm{u}_1 g_1 + \ldots + u_{k-1} g_{k-1} \tag{1.1}$$

where the binary coefficients u_i can be thought as the element of an information vector $u = [u_0, u_1, \ldots, u_{k-1}]$, which the code maps into c. Equation (1.1) 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 representations, it is always possible for a linear code to derive a representation known as systematic.

Definition 1.1.3 (Systematic Code) A linear error correcting code $\mathcal{C}(n,k)$ is said to be in a systematic form, or 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 $s = Hx^T$ 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 can be shown 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. 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 matrix H as $\left[P^T | I_r\right]$, where ^{T} denotes transposition, 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 $G = \left[I_k | \left(B^{-1}A\right)^T\right]$.

1.1.2 Quasi-cyclic codes and circulant matrices

A quasi-cyclic (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 [39].
LEDApkc 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 $v \times v$ circulant matrix A has the following form

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{v-1} \\ a_{v-1} & a_0 & a_1 & \cdots & a_{v-2} \\ a_{v-2} & a_{v-1} & a_0 & \cdots & a_{v-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_0 \end{bmatrix}. \tag{1.2}$$

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

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 eq. (1.3), any binary circulant matrix is associated to a polynomial in the variable x having coefficients over $\mathbb{F}_2$ which 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 eq. (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 LEDApkc primitives, as computing the inverse of $v \times v$ circulant matrices is required. However, a proper selection of the size of a circulant matrix v, allows to easily generate invertible instances of it as described in the following.

1.1.3 Polynomial inversion in a finite field

To provide efficient execution for the LEDApkc 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 a finite field, with q a prime power and m a positive integer; given an element $\alpha \in \mathbb{F}_{q^m}$, the following propositions hold [40]:

(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 1.1.4 Let n be a positive integer and q a prime power such that $\texttt{gcd}(n, q) = 1$. 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 [40].

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 modulo $q^m - 1$ 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 modulo n, the polynomial $x^n - 1 \in \mathbb{F}_q[x]$ can be factored as the product of monic irreducible polynomials over $\mathbb{F}_q$:

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

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 modulo 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 modulo p.
If 2 is a primitive element of $\mathbb{Z}_p$, its order, $\mathtt{ord}_p(2)$, is equal to the order of the (cyclic) multiplicative group of the field, i.e., $\mathtt{ord}_p(2) = |(\mathbb{Z}_p \setminus \{0\}, \cdot)| = p - 1$ 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 $\mathtt{ord}_p(2) = p - 1$. Let $g(x)$ be a binary polynomial in $\mathbb{F}_2[x]/\langle x^p + 1\rangle$, with $\mathtt{deg}(g(x)) > 0$.
$g(x)$ 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 $\mathtt{gcd}(g(x), x^p + 1) = \mathtt{gcd}(g(x), (x + 1)\Phi(x))$.
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 $\mathtt{ord}_p(2) = p - 1$, 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 $\mathtt{deg}(g(x)) + \mathtt{deg}(h(x)) = p - 1$, while $\mathtt{deg}(g(x)) \leq p - 1$, which in turn leaves $\mathtt{deg}(h(x)) = 0$ 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 $\mathtt{gcd}(g(x), x^p + 1) = \mathtt{gcd}(g(x), (x + 1)\Phi(x)) = 1$ 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 $\mathtt{deg}(g(x)) > 0$, 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 $\mathtt{gcd}(g(x), x^p + 1) = \mathtt{gcd}(g(x), (x + 1)\Phi(x)) = 1$, which in turn means that $\mathtt{gcd}(g(x), x + 1) = 1$ and $\mathtt{gcd}(g(x), \Phi(x)) = 1$ 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. ■

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

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 column 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 [26]. 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 the presence of a small number of short cycles in it 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 [16]. 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 main observation of the BF decoders starts from 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 LEDApkc.

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 1.1.1, 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 1.1.1 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. A possible criterion, yielding very good error correction performances at the cost of a lower computational efficiency is to flip all the bits which are involved in the maximum number of unsatisfied parity checks, computed at line 11. 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 1.1.1 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,

Algorithm 1.1.1: 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 an $d_v \times n_0$ 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 (each 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 pn0
2  syndrome ← s                                              // bitvector with size pr0
3  iterationCounter ← 0     // scalar variable denoting the number of iterations
4  repeat
5     unsatParityChecks ← 0_{1×pr0} // counters of unsatisfied parity checks
6     for i = 0 to pn0 − 1 do
7        for j = 0 to dv − 1 do
8           assertedHbitPosition ← (i + Hsparse[j][i]) mod p + p · ⌊Hsparse[j][i] div p⌋
9           if syndrome[assertedHbitPosition] = 1 then
10             unsatParityChecks[i] ← 1 + unsatParityChecks[i]
11    maxUPC ← MAX(unsatParityChecks)
12    for i = 0 to pn0 − 1 do
13       if unsatParityChecks[i] = maxUPC then
14          BitToggle(codeword[i])                                  // codeword update
15          for j = 0 to dv − 1 do
16             assertedHbitPos ← (i + Hsparse[j][i]) mod p + p · ⌊Hsparse[j][i] div p⌋
17             BitToggle(syndrome[assertedHbitPos])
18 until syndrome ≠ 0_{1×pr0}  AND  iter < imax
19 if syndrome = 0_{1×pr0} then
20    return codeword, true
21 return codeword, false
```

or the maximum number of iterations has been reached, Algorithm 1.1.1 returns its best estimate for the corrected codeword, together with the outcome of the decoding procedure (lines 19–21).

1.1.5 McEliece cryptosystem

The McEliece cryptosystem is a public-key cryptosystem proposed by Robert McEliece in 1978 [28] 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 main stages of the McEliece cryptosystem are described next.

Key generation. In order to receive encrypted messages, Bob randomly chooses a secret binary linear block code $\mathcal{C}(n,k)$, with codeword length n, information word length k, and generation matrix $G_{k\times n}$ able to correct t or less bit errors. The $k \times n$ binary generator matrix G of the secret code (allowing efficient decoding) is used as part of the private key.
Bob also chooses, as part of the private key, other two secret matrices: a dense $k \times k$ non-singular binary scrambling matrix S, and an $n \times n$ permutation matrix P.
Bob's public key is then computed as

$$G' = SGP \tag{1.5}$$

which is the generator matrix of a public code being permutation-equivalent to the secret one. Such a code has the same size and correction capability of the private code.

Encryption. Let us suppose that Alice wishes to send a secret $1 \times k$ binary string u to Bob. She gathers Bob's public key G' from a public repository and computes the encrypted version of u as

$$x = uG' + e = \ c + e \tag{1.6}$$

where e is a random binary $1 \times n$ error vector with length n and weight t, generated by Alice.

Decryption. In order to decrypt the $1 \times n$ binary vector x, obtained as in (1.6), Bob computes

$$x' = xP^{-1}$$

where $P^{-1} = P^T$ is the inverse of the permutation matrix P. Based on eq. (1.5) and eq. (1.6), we have:

$$x' = uSG + eP^{-1}$$

Hence, x' is a codeword of the secret code affected by the error vector $e' = eP^{-1}$ of weight t. So Bob can correct the error vector e' and recover $u' = uS$, from which u is then obtained through multiplication by S^{-1}.

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, it is guaranteed that Bob is able to correct e', and hence the Decryption Failure Rate (DFR) is zero.

It is also worth noticing that the original McEliece cryptosystem only provides indistinguishability under chosen plaintext attack (IND-CPA). Suitable conversions of the cryptosystem must be exploited in order to achieve IND-CCA2. When these conversions are used, some constraints on the public code can be relaxed. For example, in such a case the private and the public codes may even coincide, thus avoiding the use of S and P, on condition that the public representation of the code does not allow efficient decoding.

1.2 The LEDApkc cryptosystem

The LEDApkc cryptosystem is derived from the McEliece cryptosystem with the following main differences:

- Non-algebraic QC-LDPC codes are used as private codes. In particular, LEDApkc hinges on a family of QC-LDPC codes, $\mathcal{C}(n,k)$ with $n = pn_0$, $k = p(n_0 - 1)$, having a (small) basic block length n_0, and a single $p \times p$ redundancy block (i.e., $r = n - k = pr_0 = p$, $r_0 = 1$) in each codeword, where p is an odd prime integer [1, 2].
- The public code is neither coincident with nor equivalent to the private code.
- Suitably designed iterative non-bounded-distance decoding algorithms are used.

The motivation of using QC-LDPC codes as private codes is in the fact that these codes are known to achieve important reductions in the public key size when used in this context [1, 29]. However, when LDPC codes are used as private codes, the public code cannot be either coincident with or equivalent to the private code. Otherwise, an attacker could search for low weight codewords in the dual of the public code and find a sparse parity-check matrix of the private code which allows efficient decoding.
For this reason, following [1], LEDApkc uses a transformation matrix Q that hides the sparse parity-check matrix H of the private code into a denser parity-check matrix $L = HQ$ of the public code. This also affects the error vector that must be corrected during decryption, which is obtained from the error vector used during encryption through multiplication by Q. However, efficient iterative decoding algorithms are proposed that exploit the knowledge of Q to achieve very good performance in terms of speed and decoding failure rate (DFR).
In fact, a well-known feature of LDPC coding is that the decoding radius of iterative decoders cannot be estimated in a deterministic way, therefore some residual DFR must be tolerated, and it must be estimated heuristically through Montecarlo simulations. This is done for all the proposed instances of LEDApkc in order to guarantee that they achieve a sufficiently low DFR.
To achieve IND-CCA2, we adopt a system conversion according to [22, 23], as explained next.

1.2.1 Description of the LEDApkc primitives

Key generation.v Consider a QC-LDPC code $\mathcal{C}(n,k)$ having $n = pn_0$, $k = p(n_0 - 1)$ and consequentially a number of redundancy symbols $r = n - k = pr_0$, with $r_0 = 1$.
Both private and public keys of the proposed primitive are formed by binary matrices, each of which composed of $p \times p$ circulant blocks.

Secret key. The secret key is defined as a $pr_0 \times pn_0$ parity-check matrix H (made of $r_0 \times n_0$, $p \times p$ circulant blocks), and by a $pn_0 \times pn_0$ transformation matrix Q (made of $n_0 \times n_0$, $p \times p$ circulant blocks). The number of asserted bits in each row/column of any circulant block (a.k.a. the weight of the block) of matrix H is denoted as d_v and its value ranges from 15 to 25. The sequence of values denoting the number asserted bits in each row/column of the first n_0 circulant blocks in Q is denoted as $\overline{m} = [m_0, m_1, \ldots, m_{n_0-1}]$, where each value m_i, $0 \leq i \leq n_0$, ranges from 1 to 10.

The private key generation process starts generating randomly the positions of the d_v asserted bits in the first row a sequence of $1 \times n_0$, $p \times p$ circulant blocks H_i, $0 \leq i \leq n_0 - 1$. Subsequently, the parity-check matrix H is defined as the concatenation of these block matrices, as follows

$$H = [H_0|H_1|H_2|\ldots|H_{n_0-1}] \tag{1.7}$$

After the parity-check matrix has been generated, the private key generation process continues through picking the n_0^2 sparse circulant blocks of the transformation matrix Q ($Q_{i,j}$, $0 \leq i,j \leq n_0 - 1$) at random. The matrix Q can be written as follows:

$$Q = \begin{bmatrix} Q_{0,0} & Q_{0,1} & \cdots & Q_{0,n_0-1} \\ Q_{1,0} & Q_{1,1} & \cdots & Q_{1,n_0-1} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{n_0-1,0} & Q_{n_0-1,1} & \cdots & Q_{n_0-1,n_0-1} \end{bmatrix} \tag{1.8}$$

The number of asserted bits in each row/column of the circulant blocks in Q is fixed according to values of the following matrix of weights

$$w(Q) = \begin{bmatrix} m_0 & m_1 & \ldots & m_{n_0-1} \\ m_{n_0-1} & m_0 & \ldots & m_{n_0-2} \\ \vdots & \vdots & \ddots & \vdots \\ m_1 & m_{n_0-1} & \ldots & m_0 \end{bmatrix}$$

where the sum of the values in each row/column is constant and equal to $m = \sum_{i=0}^{n_0-1} m_i$.

The choice of the weights $\overline{m} = [m_0, m_1, \cdots, m_{n_0-1}]$ is very important since, as we will prove in the following theorem, it decides whether Q is singular or not. In the following, we denote by $\mathbf{\Pi}\{\cdot\}$ the permanent of a matrix, and with $w(\cdot)$ the weight of a polynomial, i.e., the number of its coefficients that are nonzero.

Theorem 1.2.1 Let $p > 2$ be a prime number such that 2 is a generator of the multiplicative group of integers modulo p (i.e., $\mathtt{ord}_p(2) = p - 1$), and let Q be an $n_0 \times n_0$, $n_0 > 1$, matrix of $p \times p$ circulant blocks. If $\mathbf{\Pi}\{w(Q)\}$ is odd and $\mathbf{\Pi}\{w(Q)\} < p$, then Q is non-singular.

Proof. Since each block $Q_{i,j}$, $0 \leq i,j \leq n_0 - 1$ in Q is isomorphic to a polynomial $q_{ij}(x) \in \mathbb{F}_2[x]/\langle x^p + 1\rangle$, the determinant of the matrix Q can also be put in one/to/one correspondence with an element of $\mathbb{F}_2[x]/\langle x^p + 1\rangle$. 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 Section 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 $\mathbf{\Pi}\{w(Q)\}$. If some cancellations occur, considering Statement i) and Statement ii) above, we have that $w(d(x)) = \mathbf{\Pi}\{w(Q)\} - 2c$, where c is the overall number of cancellations. So, even when cancellations occur, $d(x)$ has odd weight only if $\mathbf{\Pi}\{w(Q)\}$ is odd. In addition, the condition $\mathbf{\Pi}\{w(Q)\} < p$ guarantees that $d(x) \neq \Phi(x)$, as $w(\Phi(x)) = p$. ■

With this result, we can guarantee that, when the sequence $\overline{m}$ is properly chosen, the matrix Q is always non-singular. As we will discuss in the following, a non-singular matrix Q is necessary for key generation to be successful.

Definition 1.2.1 The secret key (SK) of LEDApkc is defined as the pair of $p \times p$ block circulant binary matrices $\{H, Q\}$, composed by $1 \times n_0$ blocks and $n_0 \times n_0$ blocks, respectively.

Since both H and Q are formed by sparse circulant blocks, it is convenient to represent each of them through the indexes of the symbols 1 in its first row. Each index of this type requires $\lceil \log_2(p) \rceil$ bits to be stored. If we consider that the circulant blocks in any block row of Q have overall weight $m = \sum_{i=0}^{n_0-1} m_i$, then the size of SK in bits amounts to

$$S_{\text{SK}} = n_0 \left(d_v + m\right) \lceil \log_2(p) \rceil \tag{1.9}$$

In practice, the secret matrices are generated through a deterministic random bit generator (DRBG), seeded with a bit string extracted from a true random number generator (TRNG). In this case, to obtain H and Q it is sufficient to know the TRNG extracted seed of the DRBG that has been used to generate the positions of their non-null coefficients. This approach allows reducing the size of the secret key to the minimum required, as it is assumed that the TRNG output cannot be further compressed. The entity of the reduction depends on the values of the parameters involved in eq. (1.9).

Public key. Starting from H and Q, the following binary matrices are computed. First of all, the matrix L is obtained as

$$L = HQ = [L_0|L_1|L_2|\ldots|L_{n_0-1}] \tag{1.10}$$

If both d_v (the weight of each row/column of H) and $m = \sum_{i=0}^{n_0-1}$ (the sum of the weights of each row/column in the first n_0 circulant blocks in Q, previously denoted as $\overline{m} = [m_0, m_1, \ldots, m_{n_0-1}]$) are odd, then L_{n_0-1} has full-rank. In fact, $L_{n_0-1} = \sum_{i=0}^{n_0-1} H_i Q_{i,n_0-1}$ and has weight equal to $md_v - 2c$ (where c is the number of cancellations occurred in the product). If md_v is odd and $md_v < p$, L_{n_0-1} is non-singular according to Section 1.1.3.

After the computation of the multiplicative inverse of L_{n_0}, the following matrix is computed

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

where I_p denotes the $p \times p$ identity matrix.

Definition 1.2.2 The public key (PK) of LEDApkc is defined as the concatenation of $n_0 - 1$, $p \times p$ block circulant matrices as follows: $M_l = [M_0|M_1 \ \ |M_2|\ldots|M_{n_0-2}]$

From an implementation point of view, as the circulant blocks forming M_l are dense, it is convenient to store them as binary polynomials in $\mathbb{F}_2[x]/\langle x^p + 1\rangle$. The bitsize of the PK is as follows:

$$S_{\text{PK}} = (n_0 - 1)\, p \tag{1.11}$$

Encryption and decryption transformations of LEDApkc. The encryption and decryption transformations of LEDApkc are designed following the description of the previously shown McEliece encryption and decryption functions employing the mentioned family of QC-LDPC codes, with a systematic generation matrix G'. The systematic form of the generation matrix G' would easily allow any observer to recover the information word embedded in any encrypted message, without recovering the private key of the cipher. However, it is well-known that if the McEliece

encryption transformation is secured against adaptive chosen-ciphertext attacks (CCA2), through applying proper IND-CCA2-conversion schemes, then 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. The conversion scheme chosen to design the LEDApkc primitives is the γ-conversion scheme proposed by Kobara and Imai in [23]. In the following, we describe the basic encryption and decryption transformations and, subsequently, the mechanisms of the γ-conversion scheme which allow us to obtain a IND-CCA2 version of the LEDApkc primitives.

Encryption function. Given the $(n_0 - 1)$, $p \times p$ circulant blocks of a public key $M_l = [M_0|M_1|M_2|\dots|M_{n_0-2}]$, the $pk_0 \times pn_0$ systematic generation matrix employed in the LEDApkc primitive is composed as $G' = \left[I_k|M_l^T\right]$, where I_k denotes a $k \times k$ identity matrix, while ^{T} denotes the transposition operation.
Given an information word u as a $1 \times p(n_0 - 1)$ binary vector, the sender computes the $1 \times pn_0$ binary vector corresponding to the encrypted message x as follows:

$$x = uG' + e = u\left[I_k|M_l^T\right] + e \tag{1.12}$$

where e is a $1 \times pn_0$ random binary error vector and Hamming weight t, where t is the system parameter denoting the error correction capability of the QC-LDPC code.

Decryption function. Given the $1 \times pn_0$ binary vector representing an encrypted message x, to recover both the binary $1 \times pk_0$ information word u, and the binary $1 \times pn_0$ error vector e, the receiver computes the binary $p \times 1$ vector of the message syndrome employing the secret matrices H and Q included in its private key SK, as follows:

$$s^T = (HQ)x^T \tag{1.13}$$

It is worth noting that for the value of the syndrome s^T the following chain of equalities holds:

$$s^T = (HQ)x^T = (HQ)(uG' + e)^T = (HQ)e^T = H(eQ^T)^T$$

as, the product

$$(HQ)G'^T u^T = (\,(HQ)\,G'^T\,)\;u^T = 0_{p\times p(n_0-1)}\;u^T = 0_{p\times 1}$$

The computed syndrome s^T can be seen as the syndrome of the expanded $1 \times pn_0$ error vector $e' = eQ^T$ through the $p \times pn_0$ parity-check matrix H; it is thus possible to retrieve the original $1 \times pn_0$ error vector e, via the efficient decoding algorithm described in Section 1.2.2. Indeed, the syndrome decoding algorithm in Section 1.2.2 allows to recover e directly from s^T, taking into account the effect of transformation matrix Q while decoding.

After recovering e, according to eq. (1.12), Bob computes

$$x + e = u\left[I_k|M_l^T\right] \tag{1.14}$$

and recovers u by looking at the first $k = p(n_0 - 1)$ bits of the vector resulting from eq. (1.14). In case a decoding error occurs, it must be notified to the sender, who has to perform the encryption again.

LEDApkc IND-CCA2 encryption and decryption functions. To provide strong security guarantees with the proposed PKC scheme, we consider the primitive conversion for code-based cryptosystems proposed by Kobara and Imai in [23], i.e., the so called Kobara-Imai (KI) γ-construction. As it will be clear from the subsequent description, LEDApkc provides a PKC with IND-CCA2 limitedly to the secure lifetime of any key pair against reaction attacks, that however can be easily estimated.

The main intuition of the KI conversion relies on `xor`-combining the plaintext to be encrypted with the output of a DRBG, seeded with the random bit-string extracted from a TRNG. As a consequence, the obtained obfuscated plaintext can be safely enciphered as the ciphertext will appear perfectly random. In order for the IND-CCA2 property to hold, a fresh random bit-string should be available from a TRNG for each message encryption. In order to be able to successfully decrypt the plaintext, the seed of the DRBG is also enciphered alongside the message. The secret seed is not simply concatenated to obfuscated plaintext, but it is `xor`-combined with the digest computed by a hash function fed with the obfuscated plaintext. The resulting obfuscated secret seed is then prefixed to the obfuscated plaintext to compose a message which is ready to be encrypted with the McEliece encryption function previously defined. In particular, the k least significant bits of this message are employed as information word of the selected QC-LDPC code $\mathcal{C}(n,k) = \mathcal{C}(pn_0, p(n_0-1))$, while the remaining most significant bits are encoded (through a specific map called *Constant Weight Encoding* function) to obtain an n-bit error vector with a number of asserted bits equal to the correction capability of the code t. Given the information word and the error vector derived from the concatenation of the obfuscated TRNG seed and the obfuscated plaintext, the McEliece encryption function is then employed to compute the codeword representing the ciphertext.
As the 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 $l_{\mathtt{s}}$.

A pseudo-code description of the KI encryption algorithm is shown in Algorithm 1.2.1. The described procedure employs the QC-LDPC code parameters of choice ($\mathcal{C}(n,k) = \mathcal{C}(pn_0, p(n_0-1))$ able to correct up to t bit errors), and a hash function, HASH(), having a digest with a byte-aligned bit-length l_{HASH} as configuration parameters. The procedure takes as input the LEDApkc public key, PK, and a plaintext bit-string `ptx` with a bit-length $0 \leq l_{\mathtt{ptx}} \leq l_{input} - 2$, where l_{input} is specified as reported below. As an output, the LEDApkc encryption primitive returns a bit-string `ctx` with $l_{\mathtt{ctx}} = pn_0$ bits – corresponding to the $1 \times pn_0$ binary codeword computed by the QC-LDPC McEliece encryption function.

The maximum length of encipherable plaintext is bounded by $l_{input} - 2$, where l_{input} is a value which need to be properly configured, keeping into account the sequence of operations in the KI-γ scheme. The LEDApkc encryption transformation shown in Algorithm 1.2.1 starts from obtaining a fresh random seed from a TRNG, which should be long enough to match the desired security level, and derives a pseudorandom bit-string `pad` having the same length of a padded `ptx`. Subsequently, an obfuscated plaintext bit-string `obfuscatedPtx` is obtained combining the `pad` via `xor` with the `ptx`, which in turn has also been padded with a fixed-format binary string composed as a sequence of zeros enclosed between two ones (lines 2–4). The DRBG employed in the scheme is required to be a NIST standard DRBG, which leaves EC-DRBG, HMAC-DRBG and CTR-DRBG as the only

Algorithm 1.2.1: LEDApkc 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_{HASH}

Input: `ptx` plaintext bit-string, with $0 \leq l_{\texttt{ptx}} \leq l_{input} - 2$, where $l_{input} = l_{\texttt{eword}} + p(n_0 - 1) - l_{\text{HASH}}$, and $l_{\texttt{eword}} = \left\lfloor \log_2 \binom{pn_0}{t} \right\rfloor - \left(\left(p(n_0 - 1) + \left\lfloor \log_2 \binom{pn_0}{t} \right\rfloor \right) \bmod 8 \right)$. In case the actual length of the input plaintext is less than $l_{\texttt{ptx}}$, it is assumed to be padded with the fixed-format string built as a sequence of zero or more 0s enclosed by two 1s.
LEDApkc public key PK: $M_l = [M_0, \ldots, M_{n_0-2}]$, where M_i, $0 \leq i \leq n_0 - 1$ is a $p \times p$ circulant block.

Output: `ctx` ciphertext bit-string

1 **repeat**
2 `seed` $\leftarrow$ TRNG()
3 `pad` $\leftarrow$ DRBG(`seed`) `// bit-string with length` l_{input}
4 `obfuscatedPtx` $\leftarrow$ CONSTANTPAD(`ptx`, l_{input}) $\oplus$ `pad`
5 `obfuscatedSeed` $\leftarrow$ ZEROEXTEND(`seed`, l_{HASH}) $\oplus$ HASH(`obfuscatedPtx`)
6 `toencode` $\leftarrow$ `obfuscatedSeed`||`obfuscatedPtx`
7 `eword`, `iword` $\leftarrow$ SPLIT(`toencode`, $l_{\texttt{eword}}$, $l_{\texttt{iword}}$)
8 $u \leftarrow$ TOVECTOR(`iword`) `//` $1 \times p(n_0 - 1)$ `information word vector`
9 e, `encodingOk` $\leftarrow$ CONSTANTWEIGHTENCODER(`eword`) `//` $1 \times pn_0$ `error vector`
10 **until** `encodingOk` $= true$
11 $x \leftarrow u \left[I_k | M_l^T \right] + e$ `//` $1 \times pn_0$ `codeword;` I_k `is a` $k \times k$ `binary identity matrix`
12 `ctx` $\leftarrow$ TOBITSTRING(x) `// bit-string with` $l_{\texttt{ctx}} = pn_0$
13 **return** `ctx`

three possible choices. CTR-DRBG was chosen as the primitive in our implementation due to its higher throughput with respect to the other solutions, when comparing implementations with the same security level.
After deriving `obfuscatedPtx`, the said bit-string is fed to the cryptographic hash function, HASH(), and the corresponding digest is combined via `xor` with the secret `seed` obtaining an obfuscated seed value `obfuscatedSeed` (line 4). Since the bit-length of the secret seed $l_{\texttt{seed}}$ and the one of the hash digest l_{HASH} may not match, and in particular $l_{\texttt{seed}} \leq l_{\text{HASH}}$ when the hash algorithm is selected to match the same security level of the DRBG, `seed` may be extended with zeroes up to l_{HASH}.
The LEDApkc encryption proceeds to concatenate `obfuscatedSeed` and the `obfuscatedPtx` (line 5), for obtaining both the information word `iword`, that will be encoded with the public QC-LDPC code, and a pseudorandom string `eword`, which is employed to compute the random error vector e to be added to the codeword resulting from the McEliece encryption function. In particular, the information word is derived from the $k = p(n_0 - 1)$ rightmost bits of the concatenation of `obfuscatedSeed` and `obfuscatedPtx` values (lines 6–7). The remaining portion of the bit-string, `eword`, cannot be employed to build an error vector since it should be a bit-string with length of $n = pn_0$ bits and with the number of asserted bits equal to the correction capability of the code t. To address this issue, a procedure called *constant weight encoding* is employed to map the `eword` values into bit-strings with length pn_0 and a number of asserted bits equal to t. The

Algorithm 1.2.2: LEDApkc 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.
HASH: hash function with digest length in bits l_{HASH}

Input: `ctx` ciphertext bit-string.
LEDApkc private key SK: H, Q secret matrices.

Output: `ptx` plaintext bit-string

1 $x \leftarrow$ ToVector(`ctx`)
2 $s^T \leftarrow (HQ)x^T$
3 $e \leftarrow$ Q-Decode(s^T, H, Q)
4 $u \leftarrow$ ExtractMostSignificantBits($x + e, p(n_0 - 1)$)
5 `iword` $\leftarrow$ ToBitString(u)
6 `eword` $\leftarrow$ ConstantWeightDecode(e)
7 `obfuscatedSeed`, `obfuscatedPtx` $\leftarrow$ Split(`eword`||`iword`)
8 `seed` $\leftarrow$ ZeroTrim(`obfuscatedSeed` $\oplus$ Hash(`obfuscatedPtx`, $l_{\texttt{eword}}$)
9 `pad` $\leftarrow$ DRBG(`seed`)
10 `ptx` $\leftarrow$ `obfuscatedPtx` $\oplus$ `pad`
11 **return** `ptx`

constant-weight bit-strings are then employed to build the $1 \times pn_0$ binary error vector e needed to compute the $1 \times pn_0$ codeword x (lines 8–9), which is in turn converted onto a ciphertext bit-string `ctx`. The codomain of the *constant weight encoding* function (line 9) is the set of length $n = pn_0$, weight t strings, in turn stating that it is possible to encode safely as one of them a bit-string `eword` with length $l_{\texttt{eword}} = \lfloor \log_2(\binom{n_0 p}{t}) \rfloor$ bits. This amount of bits, added to the length, $l_{\texttt{iword}} = p(n_0 - 1)$, of the information word `iword` yields the reliable amount of information which can be transformed into the pair of binary vectors u, e (representing the information vector and error vector, respectively) to be fed as input to a McEliece encryption function (line 11). Given the fact that the hash digest is byte-aligned, and the plaintext `ptx` is considered to be byte-aligned too by the NIST API, we also assume that the maximum practical amount of information to be encoded (line 6–7) should be byte-aligned. To this end, we diminish the length of the bit-string `eword` by the remainder modulo 8 of the sum of $l_{\texttt{iword}}$, and the *constant weight encoding* function capacity $\lfloor \log_2 \binom{pn_0}{t} \rfloor$: $l_{\texttt{eword}} = \lfloor \log_2 \binom{pn_0}{t} \rfloor - \left(\left(p(n_0 - 1) + \lfloor \log_2 \binom{pn_0}{t} \rfloor \right) \bmod 8 \right)$. This choice of diminishing the input string of the constant weight encoding has the further advantage of allowing us to employ an approximate implementation of the said function, described at the end of this Section, which offer significant performance advantages at the cost of a small loss in capacity. We note that, should the said approximate implementation fail to encode an input (line 10), it is sufficient to restart the KI encryption process with a fresh TRNG seed.
The maximum length of the bit-string in input to the LEDApkc encryption function, denoted as l_{input}, is thus given by the sum of the actual capacity of the employed constant weight encoding, plus the size of the information word $p(n_0 - 1)$, minus the size of the hash digest, i.e.: $l_{input} = l_{\texttt{eword}} + p(n_0 - 1) - l_{\text{HASH}}$. To prevent ambiguities in the interpretation of a decoded message, the LEDApkc encryption transformation takes an input plaintext `ptx` with bit-length $0 \leq l_{\texttt{ptx}} \leq l_{input} - 2$, and always pads it with a constant bit-string taking the form 10^*1 (i.e., a sequence of one or more 0 binary digits, enclosed between two bits equal to 1) to obtain a plaintext message of maximum bit-length. The constant padding is removed when the decryption process is completed, and any

decrypted message not having the properly formatted constant appended is deemed invalid. Algorithm 1.2.2 reporting the decryption procedure performs, in reverse order, the steps dual to the ones in the encryption procedure.

Constant weight encoding/decoding. Concerning the constant weight encoding and decoding procedure, we note that a straightforward approach to compute it may be obtained considering the bit string to be encoded as a constant weight one as the binary representation of an unsigned integer, and converting such an integer in a combinatorial number system [21]. Such a conversion is guaranteed to be a bijective; however computing the said bijection to convert a binary string into an n bit long constant weight one with weight t requires a computational effort in the same range as computing the binomial $\binom{n}{t}$, which adversely impacts the performance of LEDApkc for the parameters required to provide an appropriate security level (n in the tens of thousands, t in the hundreds range).

An alternative solution relies on tackles the problem of performing the constant weight encoding of the 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 word to be obtained. This approach yields practically exploitable results when the run-length encoding known as Golomb Coding [17] 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 LEDApkc. 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 $p = \Pr\{X = 0\}$ 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 with $X = 0$ being the successful event and $X = 1$ the 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 a truncated binary, 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 at each decoding passage, as follows:

i. Compute the first estimate of d, 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 $p_{i+1} = \frac{t_i - 1}{n_i - l_i}$

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' 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. Since the described procedure achieves an efficiency in the 98%–99% range according to [34], and thus is not able to convert any $\lfloor \log_2 \binom{n}{t} \rfloor$ bit long string into a constant weight vector, we implemented the constant weight encoding procedure assuming that the string to be encoded is at least $\lfloor \log_2 \binom{n}{t} \rfloor$ bit long and the eventual extra bits required are obtained materializing zeroes on the fly. After the constant weight decoding phase, the decoded compact binary string is trimmed to the appropriate length by the receiving end, in turn eliminating possible ambiguities.

1.2.2 An efficient decoding algorithm for LEDApkc

While it is possible to perform decoding of the private QC-LDPC code in LEDApkc employing a classic BF decoder such as the one described in Algorithm 1.1.1, such a choice would not exploit to the utmost the correction power of the QC-LDPC code at hand. Indeed, the positions of the errors to be corrected are not uniformly distributed; instead they depend on the positions of the ones in Q^T, which are known to the decoder.

Starting from classical BF, we have developed an improved decoder that is specifically designed for LEDApkc, where the position of the ones in the expanded error vector e' to be corrected is 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 matrix Q to estimate with greater efficiency the locations of the bits to flip, we denote it as *Q-decoder*.

Inputs of the decoder are the syndrome s' according to eq. (1.13) and the matrices H and Q according to eq. (1.7) and eq. (1.8), respectively. Output of the decoder is a $1 \times n$ vector $\hat{e}$ or a decoding failure, where $\hat{e}$ represents the decoder estimate of the error vector e appearing in (1.13). Decoding goes through a maximum of l_{max} iterations, each iteration is fed with $s^{(l-1)}$ and $\hat{e}^{(l-1)}$, and outputs $s^{(l)}$ and $\hat{e}^{(l)}$. Initialization is performed by setting $s^{(0)} = s$ and $\hat{e}^{(0)} = 0_n$, where 0_n is the length-n vector with all-zero entries. The l-th iteration of the Q-decoder performs the following operations:

i. Compute $\Sigma^{(l)} = [\sigma_1^{(l)}, \sigma_2^{(l)}, \cdots, \sigma_n^{(l)}] = s^{(l-1)}H$ (this multiplication is performed lifting all the elements of both $s^{(l-1)}$ and H in the integer domain $\mathbb{Z}$, hence $\Sigma^{(l)}$ is a vector of integers having entries between 0 and d_v).

ii. Compute $R^{(l)} = [\rho_1^{(l)}, \rho_2^{(l)}, \cdots, \rho_n^{(l)}] = \Sigma^{(l)}Q$ (this multiplication is performed in $\mathbb{Z}$ as well).

iii. Define $b^{(l)} = \max_{j=1,2,\cdots,n} \left\{ \rho_j^{(l)} \right\}$ and $\Im^{(l)} = \left\{ v \in [1, n] |\ \rho_v^{(l)} = b^{(l)} \right\}$.

iv. Update $\hat{e}^{(l-1)}$ as

$$\hat{e}^{(l)} = \hat{e}^{(l-1)} + \sum_{v \in \Im^{(l)}} q_v,$$

where q_v is the v-th row of Q^T.

v. Update the syndrome as

$$s^{(l)} = s + \hat{e}^{(l)} H^T$$

vi. If the weight of $s^{(l)}$ is zero then stop decoding and return $\hat{e}^{(l)}$.

vii. If $l+1 \leq l_{max}$ then increment l and go back to step i), otherwise stop decoding and return a decoding failure.

As in classical BF, the first step of this algorithm computes the vector $\Sigma^{(l)}$. Each entry of this vector counts the number of unsatisfied parity-check equations corresponding to that bit position, and takes values in $[0; d_v]$. This evaluates the likelihood that the entry of e' at the same position is one. Differently from classical BF, in step ii) the correlation $R^{(l)}$ between these likelihoods and the rows of Q^T is computed. In fact, the expanded error vector $e' = eQ^T$ can be written as the sum of the rows of Q^T indexed by the support of e, that is

$$e' = \sum_{j \in \Psi\{e\}} q_j \tag{1.15}$$

where $\Psi\{e\}$ denotes the support of e.

Since both Q and e are sparse (that is, $m, t \ll n$), cancellations between ones are very unlikely. When the correlation between $\Sigma^{(l)}$ and a generic row q_v of Q^T is computed, two cases may occur:

- If $v \notin \Psi\{e\}$, then it is very likely that q_v has a very small number of common ones with all the rows of Q^T forming e', hence the correlation is small.
- If $v \in \Psi\{e\}$, then q_v is one of the rows of Q^T forming e', hence the correlation is large.

The main difference with classical BF is that, while in the latter all error positions are considered as independent, the Q-decoder exploits the correlation among expanded errors which is present in LEDApkc, due to the effect of Q^T. This allows to achieve important reductions in the number of decoding iterations. As a further advantage, this decoder allows recovering e, besides e', without the need of computing and storing the inverse of matrix Q^T.

1.2.3 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. As described in Section 1.2.2, a natural choice is to set the threshold used at the l-th iteration equal to the element with maximum value of the correlation $R^{(l)}$. 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, it has some drawbacks in terms of complexity, since the computation of the maximum correlation entails significant memory storage and some repeated operations.

Therefore, we consider a different strategy, 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. This strategy still allows to achieve a sufficiently low DFR, but within a significantly smaller number of decoding iterations.

Let us consider the l-th iteration of the Q-decoder, and denote by t_l the weight of the error vector $e^{(l)}$ and with t'_l the weight of the corresponding expanded error vector $e'^{(l)} = e^{(l)}Q^T$. Let us

introduce the following probabilities [5]

$$p_{ci}(t'_l) = \sum_{j=0,\, j \text{ odd}}^{\min\left[n_0 d_v - 1, t'_l\right]} \frac{\binom{n_0 d_v - 1}{j}\binom{n - n_0 d_v}{t'_l - j}}{\binom{n-1}{t'_l}} \tag{1.16}$$

$$p_{ic}(t'_l) = \sum_{j=0,\, j \text{ even}}^{\min\left[n_0 d_v - 1, t'_l - 1\right]} \frac{\binom{n_0 d_v - 1}{j}\binom{n - n_0 d_v}{t'_l - j - 1}}{\binom{n-1}{t'_l - 1}} \tag{1.17}$$

where:

- $p_{ci}(t'_l)$ is the probability that a codeword bit is error-free and a parity-check equation evaluates it wrongly;
- $p_{ic}(t'_l)$ is the probability that a codeword bit is in error and a parity-check equation evaluates it correctly.

In both cases, the syndrome bit is equal to 1. Thus, the probability that each syndrome bit is equal to 1 can be obtained as $p_{ic}(t'_l) + p_{ci}(t'_l)$; so, the average syndrome weight at iteration l can be computed as

$$w_s^{(l)} = E\left[wt\left\{s^{(l)}\right\}\right] = \left[p_{ic}(t'_l) + p_{ci}(t'_l)\right] p \tag{1.18}$$

where $wt\{\cdot\}$ denotes the Hamming weight. Since both the parity-check matrix and the error vector are sparse, the probability of $wt\left\{s^{(l)}\right\}$ being significantly different from $w_s^{(l)}$ is negligible.

So, (1.18) allows predicting the average syndrome weight starting from t'_l. In order to predict how t'_l varies during iterations, let us consider the i-th codeword bit and the corresponding correlation value $\rho_i^{(l)}$ at iteration l. The probability that such a codeword bit is affected by an error can be written as

$$\begin{aligned} P\left\{e_i = 1 | \rho_i^{(l)}\right\} &= \frac{P\left\{e_i = 1, \rho_i^{(l)}\right\}}{P\left\{\rho_i^{(l)}\right\}} = \\ &= \frac{P\left\{e_i = 1, \rho_i^{(l)}\right\}}{P\left\{e_i = 1, \rho_i^{(l)}\right\} + P\left\{e_i = 0, \rho_i^{(l)}\right\}} = \\ &= \left(1 + \frac{P\left\{e_i = 0, \rho_i^{(l)}\right\}}{P\left\{e_i = 1, \rho_i^{(l)}\right\}}\right)^{-1} \end{aligned} \tag{1.19}$$

where e_i is the i-th bit of the error vector used during encryption. After some calculations, we obtain

$$P\left\{e_i = 1 | \rho_i^{(l)}\right\} = \frac{1}{1 + \frac{n - t_l}{t_l}\left(\frac{p_{ci}(t_l)}{p_{ic}(t_l)}\right)^{\rho_i^{(l)}}\left(\frac{1 - p_{ci}(t_l)}{1 - p_{ic}(t_l)}\right)^{m d_v - \rho_i^{(l)}}} \tag{1.20}$$

where $p_{ci}(t_l)$ and $p_{ic}(t_l)$ 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 eq. (1.18).

The decision to flip the i-th codeword bit is taken when the following condition is fulfilled

$$P\left\{e_i = 1|\rho_i^{(l)}\right\} > (1+\Delta)P\left\{e_i = 0|\rho_i^{(l)}\right\} \tag{1.21}$$

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 $P\left\{e_i = 0|\rho_i^{(l)}\right\} = 1 - P\left\{e_i = 1|\rho_i^{(l)}\right\}$, (1.21) can be rewritten as

$$P\left\{e_i = 1|\rho_i^{(l)}\right\} > \frac{1+\Delta}{2+\Delta}. \tag{1.22}$$

$P\left\{e_i = 1|\rho_i^{(l)}\right\}$ is an increasing function of $\rho_i^{(l)}$, hence the minimum value of $\rho_i^{(l)}$ such that (1.22) is satisfied can be computed as

$$b^{(l)} = \min\left\{\rho_i^{(l)} \in [0; md_v], \ : \ P\left\{e_i = 1|\rho_i^{(l)}\right\} > \frac{1+\Delta}{2+\Delta}\right\} \tag{1.23}$$

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) and (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 biggest 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 bigger values of w_j to the smaller 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.

Chapter 2

Security analysis

2.1 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 NP (Nondeterministic Polynomial) 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 time class, BQP [9]. In 1997 Peter Shor proved that the integer factoring problem, which has its decisional version in NP, is effectively in BQP [37], 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.

LEDApkc is constructed starting from the computational problem of performing the decoding of a codeword, i.e., deriving the corresponding error vector with a bounded weight for a general linear code, which was shown to be NP-complete in [7]. Indeed, in [7] the authors show that there is no exponentially faster way to compute the error vector of a general linear code than through enumerating and testing all the possible ones, unless P=NP. While the public matrix M of the LEDApkc cryptosystem has quasi-cyclic structure, we note that no algorithms currently exist exploiting this regularity to gain an exponential advantage in decoding the corresponding code.

With this statement standing, the security analysis of LEDApkc examines and quantifies the effectiveness of the best known attacks detailing the efficiency of algorithms running on both classical and quantum computers providing non-exponential speedups over an enumerative search for the

correct error vector. We remark that currently no algorithm running on either a classical TM or a quantum TM provides an exponential speedup in solving the computational problem underlying LEDApkc compared to an exhaustive search approach.

2.2 Analysis of the algorithm with respect to known attacks

Squaring Attacks. Another potential attack to systems based on QC-LDPC codes is the one presented in [36]. This attack uses a so-called *squaring technique* to find a a low-weight error vector and thus low-weight codewords more efficiently than with a general information set decoding (ISD) algorithm. This attack, however, is applicable if and only if the size of the circulant blocks p is even. In LEDApkc p is chosen as a prime both as a conservative choice against cryptanalysis exploiting factorization of p, and gaining in terms of efficiency due to the invertibility test reported in Theorem 1.2.1.

Decoding Attacks (DA) and Key Recovery Attacks (KRA). Concerning attacks which aim at solving efficiently the decoding of a message exploiting the public code representation, a prime position is occupied by the ISD approach. The ISD approach attempts at performing the decoding of a general linear code (polynomially) more efficiently than an exhaustive search approach, and was pioneered by Prange in [32]. Subsequent improvements of Prange's algorithm were presented by Lee-Brickell [24], Leon [25] and Stern [38] improving, although polynomially on Prange's original algorithm. Among these variants, the most noteworthy one is the one by Stern [38] which is currently the one best exploiting the speedups provided by quantum computers according to [12]. In particular, a significant portion of Stern's algorithm can be solved employing Grover's algorithm [18] to cut down the running time to the square root of the one needed for a computation on a classical platform. By contrast,when considering efficient execution on classical computers the most efficient ISD turns out to be the Becker-Joux-May-Meurer (BJMM) algorithm proposed in [6] which is part of a family of recent results [8, 27, 30, 31]. As a consequence, the security levels against attackers performing a decoding attack (DA) with classical computers have been estimated by considering the work factor of the BJMM algorithm, while the security levels against quantum computer-equipped attackers were computed taking into account Stern's algorithm.

A different approach at attacking the system is the one of efficiently finding low-weight codewords in the dual of the public code of the cryptosystem. It is possible to show that the low weight codeword finding problem is equivalent to the general linear code decoding problem, thus allowing ISD to be retrofit to this task too. In the case of LEDApkc we note that the matrix $L = HQ$, which is generally sparse, is a valid parity-check matrix for the public code. Since the rows of L are sparse codewords of the code generated by M (that is, the dual code of the public code), and have weight in the order of $n_0 d_v m$, an attacker may search for them in the dual of the public code. An opponent may thus exploit an efficient algorithm for the search of low-weight codewords in linear block codes, thus performing a key recovery attack (KRA) on L, row by row. Then, because of the sparsity of L, the opponent may succeed in separating H from Q and recovering the secret key. Alternatively, the attacker may just attempt to perform the decoding action employing the entire L, which has low weight, as a parity-check matrix.

We defend LEDApkc from both DAs and KRAs employing parameters which prevent the low-weight codeword finding from succeeding given a computational power bounded by the desired security

level. To this end, we take into account the fact that the nature of the QC codes employed in LEDApkc provides a speedup by a factor $\sqrt{p}$ with respect to the running time of the ISD algorithm employed to perform a general linear code decoding [35]. Instead, when the ISD algorithm is employed with the purpose of retrieving low-weight codewords in L the speedup factor provided by the QC structure of L is p with respect to its application to a general code. Both these speedup factors are taken into account in our estimates of the security level for a given parameter set.

Quantum Stern's algorithm. Considering the fact that Stern's algorithm [38] is the one best suited for quantum computer execution, and will thus be employed to determine the parameters of LEDApkc, we briefly resume the results in [12], describing how the application of Grover's algorithm to ISD can be taken into account when computing the complexity of KRAs and DAs.

ISD is an algorithm $\mathcal{A}(\mathcal{C}(n,k),w)$ taking as input a code $\mathcal{C}(n,k)$ with length n, dimension k, and tries to find a codeword of weight w or, equivalently, an error vector with weight w given the code and the corresponding syndrome.

In LEDApkc employing the ISD to perform a general decoding will have it acting on an $n_0 p$ bits long code, with dimension $(n_0-1)p$, trying to correct t errors, while employing it to perform low-weight codeword finding is equivalent to running it on a code $n_0 p$ bits long, with dimension p, trying to correct $n_0 d_v m$ errors.

The basic structure of each ISD algorithm is essentially the same, and is based on the identification of an *information set*, that is, a set of k linearly independent columns of the generator matrix of the code. Recovering the entries of the error vector affecting this set is enough to reconstruct the whole error vector. The algorithm must be run iteratively, and each iteration has a probability of success $p_{\mathcal{A}}$. Thus, the expected number of iterations that makes the attack successful is $\frac{1}{p_{\mathcal{A}}}$. The probability $p_{\mathcal{A}}$ is obtained as the product of p_{inv} and p_e, where p_{inv} is the probability that an iteration of ISD has selected a set of k linearly independent vectors, while p_e is the probability that the error vector entries affecting the selected set can be recovered. It can be proven that p_{inv} converges to $p_{inv} \approx 0.29$ as the size of the binary matrix being inverted increases, while for p_e we have

$$p_e = \frac{\binom{w}{2m}\binom{n-w}{k-2m}\binom{2m}{m}\binom{n-k-w+2m}{l}}{4^m\binom{n}{k}\binom{n-k}{l}}$$

according to [38], where l and m are parameters which influence the complexity of the algorithm and must be optimized to minimize the value of p_e.

Taking into account the speedup following from the application of Grover's algorithm to Stern's algorithm, it follows that the algorithm is successful after performing only $\frac{\pi}{4}\sqrt{\frac{1}{p_{\mathcal{A}}}} = \frac{\pi}{4}\sqrt{\frac{1}{p_{inv}p_e}}$ iterations on average, instead of $\frac{1}{p_{inv}p_e}$. Let us define:

- c_{dec} as the cost in qubit operations of decoding the input qubits to the inputs of the classical algorithm which must be performed whenever an iteration completed on the quantum computer;
- c_{it} as the number of bit operations needed to perform an iteration of the classical Stern's algorithm;
- c_{inv} as the cost of inverting the matrix obtained with the k columns selected during the iteration; in fact, because a quantum implementation of Stern's algorithm must be performed

entirely with revertible operations, skipping an iteration is not possible, even if the selected k-columns do not correspond to an information set (i.e., they are not linearly independent).

By taking the conservative assumption that a qubit operation has the same cost of a bit operation, it is possible to express the amount of operations required to execute Stern's algorithm on a quantum computer as

$$\frac{\pi}{4}\sqrt{\frac{1}{p_{inv}p_e}}(c_{dec}+c_{inv}+c_{it}) \tag{2.1}$$

Estimating the actual value of c_{dec} can be very hard, since it depends on the size of the input given to $\mathcal{A}$. For example, some input parameters can be fixed (in this case, the number of bits needed to represent the input given to $\mathcal{A}$ decreases) but, at the same time, the value of p_e might get lower (since, in this case, we might not consider an optimal input choice). While estimates for c_{dec} have put it in the 2^{30} range [12], we conservatively consider $c_{dec}=0$. Finally, to compute the two remaining computational costs, we refer to the following expressions from [38]:

$$c_{it}=2lm\binom{k/2}{m}+2m(n-k)\binom{k/2}{m}^2 2^{-l} \tag{2.2}$$

$$c_{inv}=\frac{1}{2}(n-k)^3+k(n-k)^2 \tag{2.3}$$

BJMM algorithm complexity. As already mentioned, when only classical computers are available, the most efficient ISD algorithm turns out to be the BJMM algorithm proposed in [6]. A precise estimation of the work factor of this algorithm in the finite-length regime can be found in [20], and it has been used to compute the work factor of attacks based on ISD against the proposed instances of LEDApkc, when performed with classical computers. While the complete expression of the computational complexity of the BJMM algorithm is rather complex, we point out that a simple expression providing an approximate but fairly intuitive expression for it is reported in [11]: 2^{cw}, where $c=\log_2\frac{1}{1-\frac{k}{n}}$.

2.3 System parameters for the required security categories

Nine sets of parameters for LEDApkc are proposed, clustered into in three classes corresponding to different security guarantees. Each one of the three instances in each class correspond to a different value of n_0 $(2,3,4)$, yielding a different balance between performance and public key size. The parameters of the nine instances of LEDApkc are reported in Table 2.1 for the NIST required security categories 1, 3 and 5, respectively. We assume that the security requirements of category 2 can be fulfilled employing category 3 parameters, and the security requirements for category 4 are fulfilled by category 5 parameters. In the table, the superscript (pq) denotes that the attack work factor has been computed taking into account quantum speedups due to Grover's algorithm, while the superscript (cl) denotes that only classical computers have been considered in evaluating the attack work factor.

For each security category and considered value of n_0, computed the minimum values of d_v md_v and t which ensure the desired the security level against KRA and DA, respectively. Then we

selected a value for the circulant block size p as a prime with $\mathtt{ord}_2(p) = p - 1$ to provide efficient invertibility tests for circulant blocks. We have checked whether tm errors can be corrected by the private code through Q-decoding maintaining a sufficiently low DFR via Montecarlo simulations. In particular, we targeted 10^{-8} as an acceptable DFR. Otherwise, we have increased the value of p and repeated the procedure.

In order to obtain a preliminary estimate of the value of p to speed up the design procedure, we exploited the BF asymptotic thresholds supplied in [5]. These thresholds allow, given the code size n, dimension k and density d_v, to compute an estimate of the biggest weight of an error vector which can be corrected by the code, using a classical BF decoder.

This approach can be used also for predicting the correcting capability of LEDApkc: indeed, we have observed that, in the waterfall region of the Q-decoder, the DFR of the system, when weight-t error vectors are used, can be approximated by the one of a BF-decoder, taking as input a parity check matrix in the same form as (1.7) and having circulant blocks with weight equal to md_v. Thus, the BF threshold can be used to predict the correcting capability of a LEDApkc instance: if the theoretical threshold is below the value of t, then a bigger value of p must be chosen. We want to stress that the described DFR estimation procedure is only approximated, and Montecarlo simulations must be performed in order to evaluate the actual DFR of the system.

In order to make a conservative design choices for the parameters, we have considered some margin in the complexity estimates of the attacks, such that the actual security level for these instances is larger than the target one. This also accounts for possible (though rare) cancellations occurring in L when computed as the product of H and Q, which may yield a row weight slightly smaller than md_vn_0, influencing the resistance to KRAs. The values of d_v have been chosen greater than 15 in order to avoid codes having small minimum distances.

To ensure that both H and Q have maximum rank, we chose d_v odd and $[m_0, m_1, \cdots, m_{n_0-1}]$ have been chosen such that the value of $\mathbf{\Pi}\{w(Q)\}$ is odd and smaller than p, allowing theorem 1.2.1 to hold. Indeed since, $L = HQ$ is a valid parity-check matrix for the public code, a singular Q, may result in the rank of L being lower than p, leading to a code with a co-dimension lower than p. When multiple choices of m and d_v were possible to achieve the same security level, we have selected those with the lowest values of the product md_v, as this is known to enhance the error correcting capability of the private code.

The system parameters design procedure can thus be summarized as follows:

i. pick a desired security level SL expressed as the $\log_2$ of the number of operations to be performed and a number of circulant blocks n_0;

ii. consider the computational effort of performing a decoding via ISD and compute the minimum number of errors $\hat{t}$, in order to ensure that DAs take more than 2^{SL} operations;

iii. consider the computational effort of performing a KRA via ISD and compute the minimum value of md_v, in order to ensure that KRAs take more than 2^{SL} operations; denote this value as $\hat{d_v'}$;

iv. compute an initial value of p such that $\mathtt{ord}_p(2) = p-1$ and that the resulting code is expected to be correcting at least t errors via Q-decoder according to asymptotic estimates;

v. choose d_v as an odd number and a set of integers $[m_0, m_1, \cdots, m_{n_0-1}]$ such that $\mathbf{\Pi}\{w(Q)\}$ is

Table 2.1: Parameters for LEDApkc and estimated computational efforts to break a given instance as a function of the security category and number of circulant blocks n_0

Category	n_0	p	d_v	$[m_0, \cdots, m_{n_0-1}]$	t	$SL_{DA}^{(pq)}$	$SL_{KRA}^{(pq)}$	$SL_{DA}^{(cl)}$	$SL_{KRA}^{(cl)}$	DFR
1	2	27,779	17	[4, 3]	224	135.43	134.84	217.45	223.66	$\approx 8.3 \cdot 10^{-9}$
	3	18,701	19	[3, 2, 2]	141	135.63	133.06	216.42	219.84	$\lesssim 10^{-9}$
	4	17,027	21	[4, 1, 1, 1]	112	136.11	139.29	216.86	230.61	$\lesssim 10^{-9}$
2–3	2	57,557	17	[6, 5]	349	200.47	204.84	341.52	358.16	$\lesssim 10^{-8}$
	3	41,507	19	[3, 4, 4]	220	200.44	200.95	341.61	351.57	$\lesssim 10^{-8}$
	4	35,027	17	[4, 3, 3, 3]	175	200.41	201.40	343.36	351.96	$\lesssim 10^{-8}$
4–5	2	99,053	19	[7, 6]	474	265.38	267.00	467.24	478.67	$\lesssim 10^{-8}$
	3	72,019	19	[7, 4, 4]	301	265.70	270.18	471.67	484.48	$\lesssim 10^{-8}$
	4	60,509	23	[4, 3, 3, 3]	239	265.48	268.03	473.38	480.73	$\lesssim 10^{-8}$

odd and smaller than p and $md_v \geq \hat{d}'_v$. If more than a solution is possible pick the one with md_v closest to $\hat{d}'_v$;

vi. simulate the DFR of the code; if it is not sufficiently low, restart from (iv) and choose a larger value of p.

In the steps (ii) and (iii) a temporary value of p is used to compute the work factor of the attacks, which however do not exhibit a significant dependence on p (and so, on n). Nevertheless, at the end of the design process, it is necessary to verify that the final value of p actually yields a work factor of both DAs and KRAs above the target security level.

Table 2.1 reports the values of the parameters derived with the aforementioned procedure for the nine instances of LEDApkc. The DFR were estimated through Montecarlo simulations: for all the parameter sets belonging to categories 3 and category 5 we computed 10^8 decoding actions without encountering a single decoding error. For the parameter set corresponding to category 1, with $n_0 = 2$ circulant blocks we obtained an estimate of the DFR of $\approx 8.3 \cdot 10^{-9}$, having encountered 20 decoding errors over $2.394 \cdot 10^9$ decoding actions. For the parameters corresponding to category 1 and $n0 \in \{3, 4\}$ we encountered no decoding errors during the computation of 10^9 decoding actions.

2.4 Reaction attacks and lifetime of a LEDApkc keypair

In addition to the proper sizing of the parameters of LEDApkc so that it withstands the aforementioned attacks, a last concern should be taken into account concerning the lifetime of a LEDApkc keypair. Indeed, whenever an attacker may gain access to a decryption oracle to which he may pose an unlimited amount of queries, the so-called *reaction attack* become applicable. Reaction attacks recover the secret key by exploiting the inherent non-zero DFR of QC-LDPC codes [15]. 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 in 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 status of the decoding (failure-non 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 [15], and provide an estimation of the number of ciphertexts required to recover the secret key of a LEDApkc instance. This analysis provides a simple criterion for estimating a secure lifetime of each keypair, which must be renewed after a fixed number of decoding failures is encountered.

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:

$$DS(v) = \{\min\{|p_i - p_j|, p - |p_i - p_j|\}, \quad p_i, p_j \in \Psi_v\} \tag{2.4}$$

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 shown in [15], it is possible to reconstruct a vector v once its distance spectrum and number of set symbols is known.

The ultimate goal of a reaction attack is the reconstruction of the matrix $\tilde{H} = HQ^T$, which is a sparse parity-check matrix for the public code and can be used to recover and correct the error vector e. In order to accomplish this task, it is necessary for the attacker to distinguish distances belonging to $DS_H = \bigcup_{i=0}^{n_0-1} DS(H_i)$ from those belonging to $DS_Q = \bigcup_{i=0}^{n_0-1} DS(Q_{0i})$. Since 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.

In these attacks, Eve generates a large number of valid plaintexts/ciphertexts pairs and sends the ciphertexts to Bob, asking for decryption. Then, the analysis on Bob's reactions (in terms of decoding success or failure) is exploited by Eve to recover DS_H and DS_Q. It is useful to decompose e into p-bit blocks, i.e., $e = [e_0, e_1, \cdots, e_{n_0-1}]$, where each block e_i has length p. Eve creates four integer vector of length $p/2$: a, b, v and u and initializes them to zero. Then, Eve generates T plaintext/ciphertext pairs and sends the ciphertexts to Bob for decryption. For each decrypted ciphertext, Eve goes through the following steps:

i. compute the distance spectra of all the error vectors $DS(e_0)$, $DS(e_1)$, $\cdots$, $DS(e_{n_0-1})$;

ii. if a distance d is contained in at least one spectrum $DS(e_i)$, increment $b(d)$ by 1;

iii. if a distance d is contained in at least one spectrum $DS(e_i)$, and there is a decoding failure, increment $a(d)$ by 1;

iv. if a distance d is contained in the spectrum of the first error vector $DS(e_0)$, increment $v(d)$ by 1;

v. if a distance d is contained in the spectrum of the first error vector $DS(e_0)$ and there is a decoding failure, increment $u(d)$ by 1.

After this procedure has been performed for all the T ciphertexts, the four vectors are used to obtain information about DS_H and DS_Q as follows. Then the ratio $p(d) = \frac{a(d)}{b(d)}$ is used to estimate the presence of a distance in: *i*) neither H nor Q, or *ii*) H only, or *iii*) Q only, or *iv*) in both H and Q. The ratio $p_Q(d) = \frac{u(d)}{v(d)}$, instead, is used to detect if a distance included in Q is included in its first row of circulant blocks or in other block rows of Q.

In order to provide an example of the use of $p(d)$, let us consider the case of a code with parameters $n_0 = 2$, $p = 2003$, $d_v = 11$, $t = 33$, $[m_0, m_1] = [3, 2]$; in Fig. 2.1 we report the values of $p(d)$, for each $d \in [1, \lfloor p/2 \rfloor]$, after the analysis of $T = 4 \cdot 10^7$ ciphertexts.

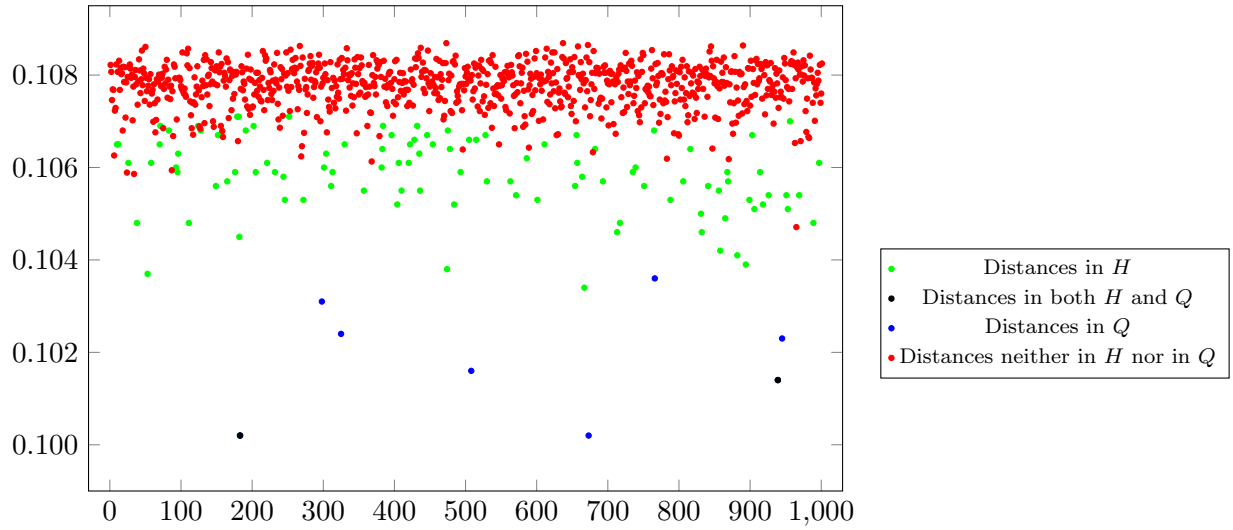


Figure 2.1: Example of distribution of $p(d)$ after collection of $T = 4 \cdot 10^7$ ciphertexts

As we can see from the figure, the estimated $p(d)$ values follow different distributions, depending on the presence of the corresponding distances in the spectrum of H or Q (or both of them). Hence, a threshold criterion can be used to assign each distance to the corresponding subset. Once this analysis has been performed, Eve knows some candidates for the blocks of H and Q. Then, by using the ratio $p_Q(d)$ she can identify distances coming from the first row of blocks of Q, and use them jointly with the public key in order to obtain $\tilde{H}$. After that, the attack can be considered successfully completed.

A critical parameter for this attack is T, which is the number of collected ciphertexts: if T is not sufficiently large, the estimates $p(d)$ are not statistically reliable and the distance spectrum analysis is inaccurate. Every time there is a decoding error, the vectors a and u are updated; the bigger the values of a and u, the more reliable the estimates, since the analysis is based on a larger number of decoding errors. We can study the mean values of a and u in order to estimate the effectiveness of the attack. After observing T ciphertexts, the average number of updates is $T \cdot \mathrm{DFR}$. We must take into account that each error vector contains only a fraction of the distances in the spectrum, so it contributes only to the update of a portion of a and u. Let us focus on a distance d, and denote as $p_{a,d}$ the probability that at least one of $DS(e_i)$ contains d, and as $p_{u,d}$ the probability that $DS(e_0)$ contains d. Then, the average values of $a(d)$ and $u(d)$ can be estimated as:

$$E\left[a(d)\right] = T \cdot \mathrm{DFR} \cdot p_{a,d} \tag{2.5}$$

$$E\left[u(d)\right] = T \cdot \mathrm{DFR} \cdot p_{u,d} \tag{2.6}$$

A simple expression for $p_{u,d}$ can be obtained as follows. The p-bit block e_0 may have weight between 0 and t. Let us suppose that its weight is t_p, which occurs with probability

$$p_{t_p} = \frac{\binom{p}{t_p}\binom{n-p}{t-t_p}}{\binom{n}{t_p}} \tag{2.7}$$

In this case, there are $N_{t_p} = \binom{t_p}{2}$ couples of ones in e_0 producing distances in the corresponding spectrum. We can assume that such distances are uniformly distributed and uncorrelated. Thus, the probability that a distance d is included in e_0 can be estimated as

$$p_{u,d}(t_p) = 1 - \frac{\binom{\lfloor p/2 \rfloor + N_{t_p} - 2}{N_{t_p}}}{\binom{\lfloor p/2 \rfloor + N_{t_p} - 1}{N_{t_p}}} = \frac{N_{t_p}}{\lfloor p/2 \rfloor + N_{t_p} - 1} \tag{2.8}$$

Then, we have:

$$p_{u,d} = \sum_{t_p=0}^{t} p_{t_p} p_{u,d}(t_p) = \sum_{t_p=0}^{t} \frac{\binom{p}{t_p}\binom{n-p}{t-t_p}}{\binom{n}{t_p}} \cdot \frac{N_{t_p}}{\lfloor p/2 \rfloor + N_{t_p} - 1} \tag{2.9}$$

Since e is made of n_0 blocks, we have:

$$p_{a,d} = 1 - (1 - p_{u,d})^{n_0} \tag{2.10}$$

We have verified through numerical simulations that (2.9) and (2.10) provide accurate approximations of the actual values of these probabilities (despite the fact that some couples of ones are indeed correlated has been neglected). Let us define $DS_U = DS_H \bigcup DS_Q$. The estimation phase of the attack can be modeled as the solution of the following two problems:

Problem 1: *for each d s.t. $1 \leq d \leq \lfloor p/2 \rfloor$, decide whether $d \in DS_U$ or not.*

Problem 2: *for each $d \in DS_U$, decide which one of the following statements is true:*

i. $(d \in DS_H) \wedge (d \notin DS_Q)$;

ii. $(d \notin DS_H) \wedge (d \in DS_Q)$;

iii. $(d \in DS_H) \wedge (d \in DS_Q)$.

Since Problem 2 can be solved only if Problem 1 has been solved, its solution cannot be easier than that of Problem 1. In order to make a conservative analysis of the system security, let us focus on Problem 1 only, and evaluate the accuracy with which an attacker is able to solve it. In other terms, we are considering only a portion of the attack, since the matrix $\tilde{H}$ can be reconstructed only if DS_H and DS_Q are correctly recovered.

Let us consider several values of T, and for each of them let us obtain the estimated values of $p(d)$. For each considered value of T, there is an optimized threshold value Θ, such that if $p(d) > \Theta$, then the attacker concludes that $d \notin DS_U$. This criterion leads to splitting the distances into two subsets: the subset $\overline{DS_U^*}$, which contains the distances that are estimated as not belonging to DS_U, and the subset DS_U^*, which contains the distances that are estimated as belonging to H or Q, or both of them. For each value of T, the number of errors in the subset assignment is the total number of distances that are wrongly assigned to a subset. Then, the accuracy of the attack can be measured as

$$\eta = 1 - \frac{\left|DS_U^* \bigcap \overline{DS_U}\right| + \left|\overline{DS_U^*} \bigcap DS_U\right|}{|DS_U|} \tag{2.11}$$

where $|\cdot|$ denotes the cardinality of a set.

We can put a threshold on η, denoted as $\hat{\eta}$, to find a secure lifetime of any keypair. A possible choice is $\hat{\eta} = 0.5$, meaning that the attacker is able to estimate correctly only half of the distances

Table 2.2: Values of $p_{a,d}$ and $p_{u,d}$ for the proposed instances of LEDApkc

$\mathbf{n_0}$	$\mathbf{p}$	$\mathbf{t}$	$\mathbf{p_{u,d}}$	$\mathbf{p_{a,d}}$
2	27,779	224	$3.09 \cdot 10^{-1}$	$5.22 \cdot 10^{-1}$
3	18,701	141	$1.04 \cdot 10^{-1}$	$2.81 \cdot 10^{-1}$
4	17,027	112	$4.35 \cdot 10^{-2}$	$1.62 \cdot 10^{-1}$
2	57,557	349	$3.44 \cdot 10^{-1}$	$5.70 \cdot 10^{-1}$
3	41,507	220	$1.14 \cdot 10^{-1}$	$3.04 \cdot 10^{-1}$
4	35,027	175	$5.14 \cdot 10^{-2}$	$1.90 \cdot 10^{-1}$
2	99,053	474	$3.61 \cdot 10^{-1}$	$5.91 \cdot 10^{-1}$
3	72,019	301	$1.22 \cdot 10^{-1}$	$3.23 \cdot 10^{-1}$
4	60,509	239	$5.53 \cdot 10^{-2}$	$2.04 \cdot 10^{-1}$

included in DS_U. In this case, the other half of the distances must be guessed by the attacker, but their enumeration is unfeasible for all the proposed instances of LEDApkc.

The accuracy of the attack depends on $a(d)$, since the confidence in the estimation of $p(d) = \frac{a(d)}{b(d)}$ increases with $a(d)$. Hence, we can translate the condition $\eta > \hat{\eta}$ for considering the attack successful into a threshold condition on $a(d)$, i.e., $E[a(d)] > \hat{a}$. Then, the lifetime $\hat{T}$ of a key pair can be assumed as the maximum value of T such that $E[a(d)] \leq \hat{a}$. With a pessimistic assumption, the effect of $p_{a,d}$ can be neglected, that is, we can assume $p_{a,d} = 1$. This way, we make the conservative assumption that, upon occurrence of a decoding failure, all distances are updated. This way, we have:

$$\hat{T} = \hat{a} \cdot \mathrm{DFR}^{-1} \tag{2.12}$$

We point out that neglecting $p_{a,d}$ results in a significant underestimation of the minimum number of ciphertexts which make the attack feasible. In Table 2.2 we report the values of $p_{a,d}$ and $p_{u,d}$ for the LEDApkc instances we propose. As we can see from the table, the values of $p_{a,d}$ range from a maximum of ≈ 0.6 to a minimum of ≈ 0.2, meaning that we are not taking into account an additional factor of approximately $2 \div 5$ in the lifetime of a key pair.

Both in [15] and in our simulations, realistic values of $E[a(d)]$ which are needed to make the attack successful are in the order of 10^6. Instead, in all our simulations values of $E[a(d)]$ in the order of 10^4 always led to small values of η (in the order of 0.5 or less). Hence, by assuming $\hat{a} = 10^4$, we assure that the attack cannot be successfully performed. The estimated lifetime of a key pair can then be assumed as $\hat{T} = 10^4 \mathrm{DFR}^{-1}$.

Considering the DFR values we have obtained through numerical simulations, we can now compute the expected lifetimes of the nine LEDApkc instances listed in Table 2.1. For the parameters sets of category 1 having $n_0 = 3$ and $n_0 = 4$, the lifetime of a key pair can be estimated as $\hat{T} \gtrsim 10^{13}$ ciphertexts. For all the others parameters sets, it can be estimated as $\hat{T} \gtrsim 10^{12}$ ciphertexts. It is important to observe that this approach can be translated into practice very easily: after a keypair encounters $\hat{a}$ decoding failures, it must be discarded.

In order to verify this approach, the attack has been simulated with several choices of the system parameters. The results of this analysis are reported in Tab. 2.3. As we can see from the table, for all the considered values of $E[a(d)]$, the attack accuracy is in the order of 0.5 or less. Since

Table 2.3: Values of η, obtained by numerical simulations, for several parameter choices.

$\mathbf{n_0}$	$\mathbf{p}$	$\mathbf{d_v}$	$\mathbf{t}$	$\mathbf{m}$	**DFR**	$\mathbf{T}$	$\mathbf{E}[\mathbf{a}(\mathbf{d})]$	η
2	901	7	20	5	$3.09 \cdot 10^{-1}$	$10 \cdot 10^6$	$5.94 \cdot 10^5$	0.33
2	2,003	11	33	5	$1.36 \cdot 10^{-1}$	$4 \cdot 10^6$	$1.3 \cdot 10^5$	0.45
3	1,301	7	22	7	$1.21 \cdot 10^{-1}$	$20 \cdot 10^6$	$2.71 \cdot 10^5$	0.46
3	2,549	13	22	9	$5.52 \cdot 10^{-2}$	$30 \cdot 10^6$	$9.74 \cdot 10^4$	0.51
4	947	7	15	7	$1.01 \cdot 10^{-1}$	$30 \cdot 10^6$	$1.65 \cdot 10^5$	0.32
4	4,327	13	45	7	$4.71 \cdot 10^{-3}$	$10 \cdot 10^6$	$1.02 \cdot 10^4$	0.20

these values are all bigger than the threshold we have set, this provides further evidence that the proposed approach guarantees security against reactions attacks of the type in [15].

On the basis of the analysis we developed, fixing $\hat{a} = 10^4$ corresponds to an attack accuracy $\eta \lesssim 0.5$, which prevents successful completion of the attack. This corresponds to an average lifetime of any keypair in the order of $\hat{T} = 10^4 \mathrm{DFR}^{-1}$ decrypted ciphertexts. For the DFR achieved by the proposed LEDApkc instances, this corresponds to an average lifetime of any keypair in the order of 10^{12} to 10^{13} decrypted ciphertexts.

To put this lifetime in perspective, consider the case, optimistic for an attacker, where he is able to query the decryption oracle obtaining an answer after 40μs of decryption time (around $1000\times$ faster than the reference implementation) plus 30μs readout and transfer time (about the latency of the loopback virtual network device in Linux). To collect 10^{13} answers from the oracle, the attacker would need $7 \cdot 10^{-5} \cdot 10^{12} = 7 \cdot 10^7$s, i.e., about two years and two months of continuous interaction with the decryption oracle before a rekeying must take place.

2.5 Properties of the cryptosystem

LEDApkc is obtained from a systematic McEliece cryptosystem with QC-LDPC codes. The advantage of employing the IND-CCA2 conversion described in [23] is that it allows to reap the benefits of a systematic representation of the code, with a limited performance overhead. We note that the KI-γ conversion is proved to attain the IND-CCA2 property in the random oracle model on the ciphertexts output by the code-based cryptoscheme to which it is applied, but, as it was proposed for algebraic decoding codes, it does not take into account the possibility of a decoding failure. Despite this fact, in combination with the existence of decoding failures in QC-LDPC codes does not allow us to claim the IND-CCA2 property for all the possible decoding instances, we note that, whenever no decryption failure happens in our cryptoscheme, the security proof of [23] remains valid. Since we work to minimize the occurrence of decoding failures and we void their usefulness for practical attacks as we refresh the keypairs before an attacker may profit from the information given by a decoding failure, we maintain that no worse practical security than an actual IND-CCA2 is achieved by LEDApkc. We also note that, while the chosen KI-γ conversion is proven IND-CCA2 secure in the random oracle model only, there are IND-CCA2 conversions for code based cryptosystems which provide security in the standard model and may be fitted instead of the KI-γ in LEDApkc (e.g., [13]). However, we note that the choice of the KI-γ conversion effectively prevents partial

plaintext recovery attacks such as the ones described in [41], as the entire codeword should be decoded correctly to produce a valid plaintext.

We note that, despite the linear nature of the QC-LDPC codes, the ciphertexts of LEDApkc are not malleable since a portion of the codeword decoded by the McEliece cryptosystem during the decryption primitive of LEDApkc is employed as an input to a cryptographically sound hash function. As a consequence, unless the said function is vulnerable to preimage attacks, an attacker will not be able to control at his will the output of the said function, even if given full control of the input.

Chapter 3

Implementation strategies and performance analysis

3.1 Procedural description of the LEDApkc primitives

To the end of providing an efficient implementation of LEDApkc, we represented each circulant block as a polynomial in $\mathbb{F}_2[x]/\langle x^p+1\rangle$ thanks to the isomorphism previously described. Consequentially, all the involved block circulant matrices are represented as matrices of polynomials in $\mathbb{F}_2[x]/\langle x^p+1\rangle$. The polynomials are materialized employing a bit-packed form of their binary coefficients in all the cases where number of non null coefficients is high. In case a polynomial has a low number of non null coefficients with respect to the maximum possible, i.e., its corresponding circulant matrix is sparse, we represent it materializing only the positions of the one coefficients as integers.

The LEDApkc key generation algorithm is reported in Algorithm 3.1.1, which takes as input the QC-LDPC code parameters and yields a private- and public-key pair. The first operation performed by the algorithm is the extraction of a private key SK as a random value, `rndPrivateMatricesSeed` generated from a TRNG (line 2 in Algorithm 3.1.1) and long enough to provide the desired security margin when deriving the secret matrices H and Q. The approach adopted to generate both the $1 \times n_0$ block matrix H and the $n_0 \times n_0$ block matrix Q is to expand the value `rndPrivateMatricesSeed` employing the NIST provided seed expander built on AES-256-CTR to draw random position for the asserted coefficients of the polynomials (blocks) of the matrices (line 2 in Algorithm 3.1.1). In case a duplicate position is drawn, it is discarded and a fresh position is drawn anew. The weights of the blocks of Q are designed in such a fashion that it is always invertible (see Section 1.2). We evaluated that repeating this generation process does not have a significant impact on the decryption phase, and thus opted to store only the value of `rndPrivateMatricesSeed` as the cipher private key SK (line 11 in Algorithm 3.1.1). Indeed, the size of H and Q during the computation of the decryption algorithm is still rather small, as their sparsity allows for a compact representation in memory, where only the position of the one-valued coefficients are materialized. Given the bit-sizes of the parameters of the cryptosystem, the positions can be materialized as either 16-bit or 32-bit integers depending on the chosen values for n_0 and p. The next step of the key generation algorithm is to compute the $1 \times n_0$ block matrix $L = HQ = [L_0, L_1, \ldots, L_{n_0-1}]$ (lines 3–6 in Algorithm 3.1.1). We recall that, given the choice of d_v odd, and $\sum_{i=0}^{n_0-1} m_i$ odd, the invertibility of the last block of L, L_{n_0-1} is guaranteed a priori (see Section 1.2). Therefore,

Algorithm 3.1.1: LEDApkc Keygen

Input: p, n_0, n, k: QC-LDPC code parameters, where p denotes a circulant block size (in bit), and n_0 denotes the number of circulant blocks of the $1 \times n_0$ parity-check matrix of the code. $n = pn_0$ (bit) denotes the codeword size, while $k = p(n_0 - 1)$ (bit) denotes the information word size.
d_v: odd weight of each circulant block of the parity-check matrix $H = [H_0 \mid H_1 \mid \ldots \mid H_{n_0-1}]$ to be generated
$[m_0, \ldots, m_{n_0-1}]$: weight of each block of the first row of the $n_0 \times n_0$ circulant block matrix $Q = \begin{bmatrix} Q_{0,0} & \cdots & Q_{0,n_0-1} \\ \vdots & \ddots & \vdots \\ Q_{n_0-1,0} & \cdots & Q_{n_0-1,n_0-1} \end{bmatrix}$ to be generated – with $\left(\sum_{i=0}^{n_0-1} m_i \right)$ odd

Output: (SK, PK) generated private-key/public-key pair

1 `rndPrivateMatricesSeed` $\leftarrow$ TRNG()
2 $H, Q \leftarrow$ GENERATEHQ($n_0, d_v, [m_0, \ldots, m_{n_0-1}]$, `rndPrivateMatricesSeed`)
3 **for** $i = 0$ **to** $n_0 - 1$ **do**
4 $\quad L_i \leftarrow 0$ `// null polynomial in` $\mathbb{F}_2[x]/\langle x^p + 1 \rangle$
5 $\quad$ **for** $j = 0$ **to** $n_0 - 1$ **do**
6 $\quad\quad L_i \leftarrow L_i + H_j Q_{j,i}$ `// polynomial mul. and add. in` $\mathbb{F}_2[x]/\langle x^p + 1 \rangle$
7 `LInv` $\leftarrow$ COMPUTEINVERSE(L_{n_0-1}) `// multiplicative inverse in` $\mathbb{F}_2[x]/\langle x^p + 1 \rangle$
8 **for** $i = 0$ **to** $n_0 - 2$ **do**
9 $\quad M_i \leftarrow$ `LInv` L_i `// polynomial multiplication in` $\mathbb{F}_2[x]/\langle x^p + 1 \rangle$
10 PK $\leftarrow [M_0 \mid \ldots \mid M_{n_0-2}]$
11 SK $\leftarrow$ `rndPrivateMatricesSeed`
12 **return** (SK, PK)

the computation of `LInv` $= L_{n_0-1}^{-1}$ is performed with a single call to the polynomial inversion algorithm (line 7 in Algorithm 3.1.1). Finally, the public key PK is generated as a $1 \times n_0 - 1$ block matrix, $[M_0 \mid \ldots \mid M_{n_0-2}]$, through multiplying `LInv` by all-but-the-last blocks of L (lines 8–9 in Algorithm 3.1.1). The last multiplication can be avoided as it will yield the identity matrix which is thus not stored in the PK (line 10 in Algorithm 3.1.1).

The procedural description of the LEDApkc encryption transformation has been shown in Section 1.2.1.

The procedural details about the LEDApkc decryption transformation have also been reported in Section 1.2.1, expect for the fact that the actual implementation of the primitive, takes as input the secret SK in the form of the seed extracted from a TRNG during the keypair generation, instead of the values of the secret matrices H, Q. The decryption starts by computing the $p \times 1$ syndrome, s^T, of the $1 \times pn_0$ error vector $e' = eQ^T$ (see Section 1.2.1, and line 2 in Algorithm 1.2.2). This syndrome corresponds to an error vector e' with weight $\approx m\,t$ and is decoded efficiently via the Q-DECODER strategy described in Section 1.2.2.

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 bit flipping algorithm does (lines $5-9$ in Algorithm 3.1.2). 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

Algorithm 3.1.2: Qdecode

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 j
16                 qrow[k] ← Qtr[i][k] − (Htr[i][k] mod p) + ((j + 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
```

Table 3.1: Running times for key generation, encryption and decryption as a function of the chosen category and number of circulant blocks n_0 on an AMD Ryzen 5 1600 CPU at 3.2 GHz

Category	n_0	KeyGen (ms)	Encrypt (ms)	Decrypt (ms)
1	2	45.30 (± 1.69)	3.11 (± 0.06)	20.87 (± 0.65)
	3	20.96 (± 0.23)	3.10 (± 0.06)	25.18 (± 2.18)
	4	17.99 (± 0.22)	3.94 (± 0.08)	28.30 (± 0.80)
2–3	2	198.49 (± 1.41)	12.06 (± 0.18)	62.55 (± 4.57)
	3	100.39 (± 0.57)	13.06 (± 0.15)	57.58 (± 2.69)
	4	72.78 (± 0.31)	14.18 (± 0.22)	59.75 (± 1.91)
4–5	2	558.84 (± 3.41)	33.96 (± 0.21)	115.36 (± 4.08)
	3	298.91 (± 4.18)	37.28 (± 0.61)	116.93 (± 5.07)
	4	208.90 (± 0.71)	39.85 (± 0.25)	157.23 (± 4.18)

denoted as `Htr` in the algorithm. This, in turn, allows to reduce the number of iterations of the innermost loop of the parity check computation (lines 7–9 in Algorithm 3.1.2) from $\frac{n_0 p}{\texttt{machine\_word}}$ to d_v. For example, considering the case of $n_0 = 2, p = 25931$ on the NIST reference platform ($\texttt{machine\_word} = 64$) the number of iterations drops from 811 to 17.

The differentiating point between the classical bit flipping algorithm and the Q-decoding concerns how the bits of the codeword being decoded are selected for flipping. Indeed, while employing a classical bit flipping algorithm would flip all the positions in e' having the highest number of unsatisfied parity checks, the Q-decoder exploits the knowledge of the secret matrix Q^T to estimate if a bit flip should be performed. To this end, the Q-decoding computes, for each bit of e being decoded (lines 12–13 in Algorithm 3.1.2) 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 3.1.2) 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 3.1.2). 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 $\bar{\texttt{w}}$ among the ones in the lookup table, which does not exceed the one of the syndrome (line 10 in Algorithm 3.1.2). The threshold for the similarity is selected as the one paired to $\bar{\texttt{w}}$ in `LutS` (line 11 in Algorithm 3.1.2).

3.2 Benchmarks on a NIST compliant platform

We provide the results of a set of execution time benchmarks performed on the reference implementation provided in the submission package. Currently, no platform specific optimizations are in place, thus we expect these results to be quite consistent across different platforms.

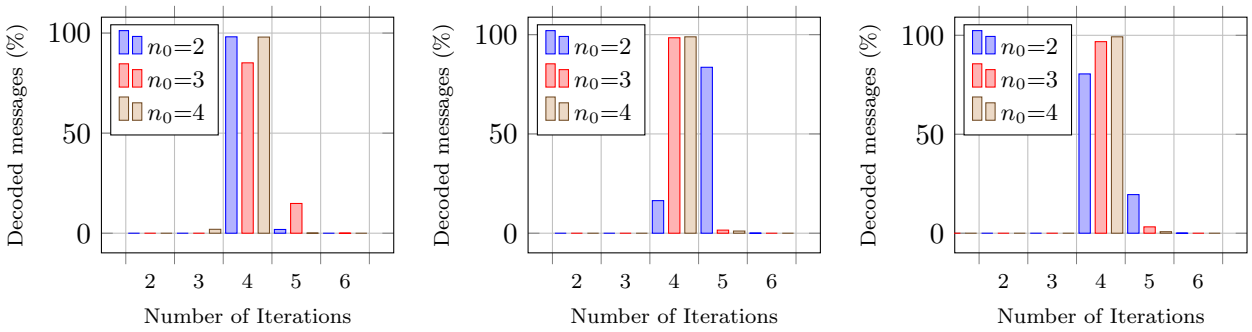
The results were obtained measuring the required time for key generation, encryption and decryption as a function of the chosen security category and number of circulant blocks n_0. The measurements reported are obtained as the average of 100 executions of the reference implementa-

Table 3.2: Sizes of the keypair, plaintext and ciphertext as a function of the chosen category and number of circulant blocks n_0

Category	n_0	Private Key Size (B)		Public Key size (B)	Max Plaintext size (B)	Ciphertext size (B)
		At rest	In memory			
1	2	24	668	3,480	3,690	6,960
	3	24	844	4,688	4,813	7,032
	4	24	1,036	6,408	6,496	8,544
2–3	2	32	972	7,200	7,558	14,400
	3	32	1,196	10,384	10,608	15,576
	4	32	1,364	13,152	13,320	17,536
4–5	2	40	1,244	12,384	12,897	24,768
	3	40	1,548	18,016	18,336	27,024
	4	40	1,772	22,704	22,955	30,272

tion compiled with `gcc` 6.3.0 from Debian 9 amd64. Given the NIST requirement on the reference computing platform (an Intel x86_64 CPU) we instructed `gcc` to employ the most basic instruction set among the ones fitting the description (`-march=nocona` option). The generated binaries were run on an AMD Ryzen 5 1600 CPU at 3.2 GHz, locking the frequency scaling to the top frequency. Table 3.1 reports the running times measured employing the `clock_gettime` primitive, selecting the `CLOCK_PROCESS_CPUTIME_ID` as the timer of choice, obtaining the CPU time taken by the process. As it can be noticed, the most computationally demanding primitive is the key generation, which has more than 80% of its computation time taken by the execution of a single modular inverse in $\mathbb{F}_2[x]/\langle x^p+1\rangle$ required to obtain the value of $L_{n_0-1}^{-1}$. The encryption primitive is the fastest among all, and its computation time is dominated ($> 93\%$) by the polynomial multiplications performing the encryption. The decryption primitive computation is dominated by the Q-decoder computation ($> 90\%$ of the time), with the remaining portion of the computation portion taken by the sparse modular multiplications which reconstruct L and the one to compute the private syndrome fed into the Q-decoder. The time required in both encryption and decryption to perform the KI-γ padding on the plaintext, and the constant weight encoding take a definitely small amount of time in both cases. Table 3.2 reports the sizes of both the keypairs and the encapsulated secrets for LEDApkc. In particular, regarding the size of the private keys of LEDApkc we report both the size of the stored private key, i.e. the size of the `rndPrivateMatricesSeed` extracted from the system TRNG, and the required amount of main memory to store the expanded key during the decryption phase. We note that the private key sizes are the minimum possible, as the `rndPrivateMatricesSeed` extracted from the system TRNG should be incompressible. We employ as a `rndPrivateMatricesSeed` size of 192, 256 and 320 bits for security categories 1, 3, and 5, respectively, to provide a simple hedging against multi key attacks, as suggested on the NIST forum, and in the NIST frequently asked questions section of the call. We note that, for a given security category, increasing the value of n_0 enlarges the public key, as it is constituted of $(n_0-1)p$ bits. This increase in the size of the public key however, enhances the underlying code rate, in turn allowing to encrypt larger plaintexts.

Possible optimizations. Starting from the platform-agnostic reference implementation provided in this submission, a number of optimizations can be applied to improve the running time of LEDApkc. First of all, implementing a sub-quadratic multiplication for the elements of $\mathbb{F}_2[x]/\langle x^p+1\rangle$, for which the best candidate for the NIST reference platform appears to be the Toom-Cook



(a) Category 1 (b) Category 2–3 (c) Category 4–5

Figure 3.1: Percentage of decoded messages as a function of the number of iterations taken to the Q-decoder to converge

method in either its Toom-3 or its Toom-4 variant [10], is expected to reduce the time required to compute the encryption primitive quite significantly. The optimal choice of the Toom-Cook variant will also be dependent on the availability on the underlying CPU of binary polynomial multiplication instructions, also known as carryless multiplications. Indeed, such instructions, present on both modern x84_64 and ARM ISAs provide significant speedups in single-precision binary polynomial multiplication.
Providing a specialized procedure for the modular inverse in $\mathbb{F}_2[x]/\langle x^p + 1\rangle$, where both the guaranteed invertibility of the element at hand, and the low weight nature of the modulus are taken into account is expected to provide a significant speedup to the key generation phase, which is dominated by a single instance of such computation.
Finally, exploiting the presence of vector instructions to perform modular addition will also provide performance boosts, as it provides an effective way of exploiting the large amount of data parallelism present in the computational primitives employed.

3.3 Protection against side-channel attacks

The two most common side channels exploited to breach practical implementations of cryptosystems are the execution time of the primitive and the instantaneous power consumption during its computation. In particular, in [14], it was shown how a QC-LDPC code-based system can be broken by means of simple power analysis, exploiting the control-flow dependent differences of the decoding algorithm examined. Furthermore, [33] provides the first practical evidence of a side channel attack relying on differential power analysis to extract the positions of the ones in the private parity-check matrix, hence exploiting dataflow dependencies. However, linear error correcting codes are amenable to extremely efficient countermeasures against power consumption-based side channel attacks due to the linear nature of the operations involved in the decoding process. Indeed, the same [33] provided a simple, but effective, countermeasure against differential power analysis through adding a random codeword to the input vector before computing the syndrome. Such a countermeasure against side channel attacks can be readily employed in LEDApkc. However, it may be possible to employ profiled side channel attack techniques to overcome this layer of protection, and against which the linearity of error correcting codes may be again exploited to devise efficient countermeasures.

Concerning execution time side channel information leakage, the main portion of the LEDApkc decryption algorithm which is not characterized by a constant execution time is decoding. Indeed, the number of iterations made by the decoder depends on the values being processed. Willing to provide a first quantitative estimate of the information leakage stemming by such a timing variation, Figure 3.1 reports the percentage of decoding actions taking a given number of iterations to perform a correct decode action during our DFR characterization campaign, i.e., over at least 10^8 decoding actions for each category/n_0 value pair. As it can be seen, the vast majority of decoding actions are completed in the same number of iterations. To provide a quantitative estimate of the information which may be obtained, we note that, considering the number of iterations required to perform a decode action as a random variable over the integers, and considering the frequencies divided by the total amount of decoding actions as a rough estimate of the actual probabilities, we obtain that the entropy of the random variables considered is between 0.21 and 0.01 bits per symbol, which is a quite limited amount of information. Nonetheless, it is possible to achieve a constant time decoding simply modifying the algorithm so that it always run for the maximum needed amount of iterations to achieve the desired DFR. Such a choice completely eliminates the timing leakage, albeit trading it off for a performance penalty. A final note concerning the timing side channel leakage is that the Kobara-Imai decryption procedure should be implemented in such a fashion that does not leak on a timing side channel whether or not a decryption failure is due to a decoding failure.

3.4 Known Answer Test values

Known answer tests generated for 100 runs of LEDApkc 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>_<value_of_the_n0_parameter>.req`
`PQCencryptKAT_<private_key_size>_<value_of_the_n0_parameter>.rsp`

In the following we report the SHA-2-256 digests of the KAT files, as obtainable via `sha256sum` or an analogous tool.

```
0b4ca0d418899e365559f4ceb0b4abbe876e7764e543dc3228f3bf8abdf22c6c  PQCencryptKAT_24_2.req
f943af7f210654d2823fe4e29b0ec6d4a44062a357968a93a9dcefacb9f4fe20  PQCencryptKAT_24_2.rsp
0b4ca0d418899e365559f4ceb0b4abbe876e7764e543dc3228f3bf8abdf22c6c  PQCencryptKAT_24_3.req
f6348df2f5f53800a5601d62aa3290e200846a3e7fbc5f6e183c8810d12cb29d  PQCencryptKAT_24_3.rsp
0b4ca0d418899e365559f4ceb0b4abbe876e7764e543dc3228f3bf8abdf22c6c  PQCencryptKAT_24_4.req
7e869287e6392a95ca1453a2b5f9c691520759088457ce88861770c927c846ec  PQCencryptKAT_24_4.rsp
0b4ca0d418899e365559f4ceb0b4abbe876e7764e543dc3228f3bf8abdf22c6c  PQCencryptKAT_32_2.req
174b2a211e6ec450fa5d1f4818c9f078285db0f45272a9946986c9bd1c8f071e  PQCencryptKAT_32_2.rsp
0b4ca0d418899e365559f4ceb0b4abbe876e7764e543dc3228f3bf8abdf22c6c  PQCencryptKAT_32_3.req
59d285b4d270547ba654b4e0dfd5fc838a16d8f3d53a3e38eeb2e1eb3a93fc32  PQCencryptKAT_32_3.rsp
0b4ca0d418899e365559f4ceb0b4abbe876e7764e543dc3228f3bf8abdf22c6c  PQCencryptKAT_32_4.req
114cc442a861a926c46652888447953192ba7519dcef1ac8eaecb9d447231809  PQCencryptKAT_32_4.rsp
0b4ca0d418899e365559f4ceb0b4abbe876e7764e543dc3228f3bf8abdf22c6c  PQCencryptKAT_40_2.req
952099040ebbf023b696b39eef87b1adb8d99d4f61dd5794a92e183988fa4138  PQCencryptKAT_40_2.rsp
0b4ca0d418899e365559f4ceb0b4abbe876e7764e543dc3228f3bf8abdf22c6c  PQCencryptKAT_40_3.req
efb830e12b1a4de3707b1149950d81ffcb9977f0213fa4a38873349cbaf01750  PQCencryptKAT_40_3.rsp
0b4ca0d418899e365559f4ceb0b4abbe876e7764e543dc3228f3bf8abdf22c6c  PQCencryptKAT_40_4.req
c74d618812f36d28ad28dce0b0858ff2bdd8e0e96ee04e39a3b1093ee641dede  PQCencryptKAT_40_4.rsp
```

Chapter 4

Summary of advantages and limitations

+ Built on an NP-complete problem under reasonable assumptions.
+ Exploits improved BF decoders which are faster than classical BF decoders.
+ Compact keypairs (below 23 kiB at most), minimum size private keys.
+ Requires only addition and multiplication over $\mathbb{F}_2[x]$, and modular inverse over $\mathbb{F}_2[x]/\langle x^p + 1\rangle$ besides single-precision integer operations.
+ Fully patent free, self contained, public domain codebase written in ANSI-C99.
+ Easy to integrate in existing cryptographic libraries.
+ Particularly efficient to apply countermeasures against non-profiled power consumption and electromagnetic emissions side channel attacks.
− Subject to reaction attacks of the type in [15], which do not apply if the life of any keypair is limited according to a given and simple criterion, yielding a secure lifetime of any keypair in the order of 10^{12} or more decrypted ciphertexts.

Bibliography

[1] M. Baldi, M. Bianchi, and F. Chiaraluce, "Optimization of the parity-check matrix density in QC-LDPC code-based McEliece cryptosystems," in *Proc. IEEE ICC 2013 - Workshop on Information Security over Noisy and Lossy Communication Systems*, Budapest, Hungary, Jun. 2013.

[2] M. Baldi, M. Bodrato, and F. Chiaraluce, "A new analysis of the McEliece cryptosystem based on QC-LDPC codes," in *Security and Cryptography for Networks*, ser. Lecture Notes in Computer Science. Springer Verlag, 2008, vol. 5229, pp. 246–262.

[3] 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.

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

[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*, ser. Lecture Notes in Computer Science, D. Pointcheval and T. Johansson, Eds. Springer Verlag, 2012, vol. 7237, pp. 520–536.

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

[8] D. J. Bernstein, T. Lange, and C. Peters, "Smaller decoding exponents: ball-collision decoding," in *CRYPTO 2011*, ser. Lecture Notes in Computer Science. Springer Verlag, 2011, vol. 6841, pp. 743–760.

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

[10] M. Bodrato, "Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0," in *WAIFI 2007 proceedings*, ser. Lecture Notes in Computer Science, C. Carlet and B. Sunar, Eds., vol. 4547. Springer Verlag, June 2007, pp. 116–133.

[11] R. Canto Torres and N. Sendrier, *Analysis of Information Set Decoding for a Sub-linear Error Weight*. Springer International Publishing, 2016, pp. 144–161. [Online]. Available: https://doi.org/10.1007/978-3-319-29360-8_10

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

[13] N. Döttling, R. Dowsley, J. Müller-Quade, and A. C. A. Nascimento, "A CCA2 secure variant of the mceliece cryptosystem," *IEEE Trans. Information Theory*, vol. 58, no. 10, pp. 6672–6680, 2012. [Online]. Available: https://doi.org/10.1109/TIT.2012.2203582

[14] T. Fabšič, O. Gallo, and V. Hromada, "Simple power analysis attack on the QC-LDPC McEliece cryptosystem," *Tatra Mountains Mathematical Publications*, vol. 67, no. 1, pp. 85–92, Sep. 2016.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] Y. Hamdaoui and N. Sendrier, "A non asymptotic analysis of information set decoding," Cryptology ePrint Archive, Report 2013/162, 2013, https://eprint.iacr.org/2013/162.

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

[22] 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.

[23] 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

[24] P. Lee and E. Brickell, "An observation on the security of McEliece's public-key cryptosystem," in *Advances in Cryptology - EUROCRYPT 88*. Springer Verlag, 1988, vol. 330, pp. 275–280.

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

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

[27] A. May, A. Meurer, and E. Thomae, "Decoding random linear codes in $O(2^{0.054n})$," in *ASIACRYPT 2011*, ser. Lecture Notes in Computer Science. Springer Verlag, 2011, vol. 7073, pp. 107–124.

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

[29] R. Misoczki, J. P. Tillich, N. Sendrier, and P. S. L. M. Barreto, "Mdpc-mceliece: New mceliece variants from moderate density parity-check codes," in *2013 IEEE International Symposium on Information Theory*, July 2013, pp. 2069–2073.

[30] R. Niebuhr, E. Persichetti, P.-L. Cayrel, S. Bulygin, and J. Buchmann, "On lower bounds for information set decoding over f_q and on the effect of partial knowledge," *Int. J. Inf. Coding Theory*, vol. 4, no. 1, pp. 47–78, Jan. 2017.

[31] C. Peters, "Information-set decoding for linear codes over F_q," in *Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science. Springer Verlag, 2010, vol. 6061, pp. 81–94.

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

[33] M. Rossi, M. Hamburg, M. Hutter, and M. E. Marson, *A Side-Channel Assisted Cryptanalytic Attack against QcBits*. Cham: Springer International Publishing, 2017, pp. 3–23.

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

[35] ——, "Decoding one out of many," in *Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science, B.-Y. Yang, Ed. Springer Verlag, 2011, vol. 7071, pp. 51–67.

[36] M. K. Shooshtari, M. Ahmadian-Attari, T. Johansson, and M. R. Aref, "Cryptanalysis of McEliece cryptosystem variants based on quasi-cyclic low-density parity check codes," *IET Information Security*, vol. 10, no. 4, pp. 194–202, Jun. 2016.

[37] 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.

[38] J. Stern, "A method for finding codewords of small weight," in *Coding Theory and Applications*, ser. Lecture Notes in Computer Science, G. Cohen and J. Wolfmann, Eds. Springer Verlag, 1989, vol. 388, pp. 106–113.

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

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

[41] P. Zajac, "A note on CCA2-protected McEliece cryptosystem with a systematic public key," *IACR Cryptology ePrint Archive*, vol. 2014, p. 651, 2014. [Online]. Available: http://eprint.iacr.org/2014/651

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 LEDApkc, 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 LEDApkc.

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 LEDApkc, 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 LEDApkc.

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 LEDApkc, 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 LEDApkc.

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 LEDApkc, 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 LEDApkc.

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 LEDApkc, 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 LEDApkc.

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: