LEDAcrypt: QC-LDPC code-based cryptosystems with bounded decryption failure rate

Marco Baldi[1], Alessandro Barenghi[2], Franco Chiaraluce[1], Gerardo Pelosi[2], and Paolo Santini[1]

[1] Università Politecnica delle Marche, Ancona, Italy
m.baldi@univpm.it, f.chiaraluce@univpm.it, p.santini@pm.univpm.it
[2] Politecnico di Milano, Milano, Italy
alessandro.barenghi@polimi.it, gerardo.pelosi@polimi.it

Abstract. We consider the QC-LDPC code-based cryptosystems named LEDAcrypt, which are under consideration by NIST for the second round of the post-quantum cryptography standardization initiative. LEDAcrypt is the result of the merger of the key encapsulation mechanism LEDAkem and the public-key cryptosystem LEDApkc, which were submitted to the first round of the same competition. We provide a detailed quantification of the quantum and classical computational efforts needed to foil the cryptographic guarantees of these systems. To this end, we take into account the best known attacks that can be mounted against them employing both classical and quantum computers, and compare their computational complexities with the ones required to break AES, coherently with the NIST requirements. Assuming the original LEDAkem and LEDApkc parameters as a reference, we introduce an algorithmic optimization procedure to design new sets of parameters for LEDAcrypt. These novel sets match the security levels in the NIST call and make the C99 reference implementation of the systems exhibit significantly improved figures of merit, in terms of both running times and key sizes. As a further contribution, we develop a theoretical characterization of the decryption failure rate (DFR) of LEDAcrypt cryptosystems, which allows new instances of the systems with guaranteed low DFR to be designed. Such a characterization is crucial to withstand recent attacks exploiting the reactions of the legitimate recipient upon decrypting multiple ciphertexts with the same private key, and consequentially it is able to ensure a lifecycle of the corresponding key pairs which can be sufficient for the wide majority of practical purposes.

1 Introduction

In this work, we provide theoretical and implementation advancements concerning quasi-cyclic low-density parity-check (QC-LDPC) code-based cryptosystems known as LEDAcrypt [3], which are under consideration by NIST for the second round of standardization of post-quantum cryptographic systems [31]. These new systems are built upon two previous systems named LEDAkem (low density parity-check code-based key encapsulation mechanism) and LEDApkc (low-

density parity-check code-based public-key cryptosystem), which were submitted to the first round of the NIST competition.

The mathematical problem on which these systems rely is the one of decoding a random-looking linear block code. Such a problem belongs to the class of NP-complete problems [6,22], which is known to contain problems without polynomial time solution on a quantum computer. This line of research was initiated by McEliece in 1978 [27], using Goppa codes as secret codes, and Niederreiter in 1986 [32], with a first attempt of introducing generalized Reed-Solomon (GRS) codes in such framework. With the main aim of reducing the public key size, several other families of codes have been considered during years, like quasi-cyclic (QC) codes [12], low-density parity-check (LDPC) codes [30], quasi-dyadic (QD) codes [28], QC-LDPC codes [4] and quasi-cyclic moderate-density parity-check (QC-MDPC) codes [29].

The distinguishing points of the LEDAcrypt cryptosystems with respect to other code-based post-quantum cryptosystems relies on the use of QC-LDPC codes as secret codes and on an efficient decoding algorithm recently introduced for codes of this kind [2]. The two main attacks that can be mounted against these systems are a decoding attack (DA) and a key recovery attack (KRA) both exploiting information set decoding (ISD) algorithms. In addition, recent attacks based on the information leakage arising from the observation of the reaction of someone decrypting ciphertexts with the same private key have proved to be effective in reducing the lifecycle of keypairs used in LEDApkc and other code-based cryptosystems characterized by a non-zero DFR [9,10,16].

In this work, we analyze all the aforementioned attacks and show how to tune the parameter design of LEDAcrypt in order to foil them. Our contributions can be summarized as follows.

(i) A quantification of the quantum and classical computational efforts required to break the Advanced Encryption Standard (AES) is provided. We rely on typical circuit design estimates for the classical computing complexity, and on the work by Grassl et al. [14] for the quantum computing complexity.

(ii) A new algorithmic approach to the design of LEDAcrypt instances with parameters matching the NIST requirements, is introduced. The proposed approach employs finite regime estimations (as opposed to asymptotic bounds [21]) of the computational efforts required to perform ISD attacks as well as to execute an exhaustive search in the parameter space of the algorithms. The parameters designed through this method yield key sizes which are significantly smaller than the original proposal to the NIST standardization effort.

(iii) A novel, closed-form upper bound on the DFR of the LEDAcrypt instances is provided. This allows to include the DFR as constraint of the parameter design, to generate of keypairs with a sufficiently small DFR to provide security against chosen ciphertext attacks, such as reaction attacks. We also report sample sets of parameters targeting a DFR of 2^{-64} for long term keys in LEDAcrypt, as well as a DFR smaller than $2^{-\lambda}$, where λ equals 128, 192 and 256, for the NIST security categories 1, 3 and 5, respectively.

The paper is organized as follows. In Section 2 we briefly recall the LEDAcrypt systems and the relevant notation. In Section 3 we define the security level benchmarks we consider, in compliance with the NIST requirements. In Section 4 we describe the attacks we take into account in the system design. In Section 5 we describe an algorithmic procedure for the design of optimized sets of parameters for these systems. In Section 6 we present the parameter sets resulting from the algorithmic design procedure for keypairs to be used in ephemeral key encapsulation mechanism (KEM)s. In Section 7 we report parameter sets that guarantee a DFR lower than a given threshold for LEDAcrypt instances with long term keys and indistinguishability under adaptive chosen ciphertext attack (IND-CCA2). The latter are derived on the basis of a theoretical characterization of the DFR of LEDAcrypt, which is reported in Appendix A. Finally, in Section 8 we draw some conclusive remarks.

2 Preliminaries and notation

LEDAcrypt exploits a secret key (SK) formed by two binary matrices: H is the binary parity-check matrix of a secret QC-LDPC code and Q is a secret transformation matrix. The code described by H has length $n = pn_0$ and dimension $k = p(n_0 - 1)$, where p is a large prime integer and n_0 is a small integer. The matrix H is formed by a row of n_0 circulant matrices with size $p \times p$ and weight d_v. The matrix Q is formed by $n_0 \times n_0$ circulant matrices whose weights coincide with the entries of $\bar{m} = [m_0, m_1, \ldots, m_{n_0-1}]$ for the first row and with those of cyclically shifted versions of $\bar{m}$ for the subsequent rows. Both H and Q are sparse, and their product gives a sparse matrix $H' = HQ$ that is a valid parity-check matrix of the public code. Due to its sparsity, H' cannot be disclosed, thus the public key is a linearly transformed version of H' that hides its sparsity. The LEDAcrypt cryptosystems hide the LDPC structure of H' multiplying all its circulant blocks by the multiplicative inverse of the last block of H' itself, yielding the public key matrix $pk^{\texttt{Nie}}$.

Concerning the error correction capability of these codes, we recall that classical hard-decision decoding algorithms used for QC-LDPC codes are known as Bit Flipping (BF) decoders. The LEDA cryptosystems employ a different decoding strategy which, while retaining a fix point BF approach, is more efficient than the schoolbook BF. Such a procedure, known as *Q-decoder* [2], relies on the fact that the (secret) parity-check matrix $sk^{\texttt{Nie}} = H'$ is obtained as the product of two sparse matrices, i.e., $H' = HQ$, where H has size $(n - k) \times n$ and number of non-zero elements in a row equal to $d_c = n_0 d_v \ll n$, while Q has size $n \times n$ and number of non-zero elements in a column equal to $m = \sum_i m_i \ll n$. Both the BF and the Q-decoder are not bounded-distance decoders, therefore their decoding radius cannot be easily ascertained for a given, weight t, error, resulting in a non-null decoding failure rate in practical scenarios. We denote as $t \ll n$ the number of errors that can be corrected by the code defined by H' with a sufficiently high probability, and the code itself is denoted as $C(n, k, t)$.

Given a value for t, the encryption is performed encoding the secret message with the public code followed by the addition of a weight t error vector (in the McEliece setting) or mapping the secret message into a weight-t binary error vector and computing its syndrome through the public code (in the Niederreiter setting). The decryption of LEDAcrypt KEM performs syndrome decoding on the received syndrome after multiplying it by the last circulant block of $sk^{\texttt{Nie}} = H'$. The decoding retrieves the error vector e except for the cases where a decoding failure takes place.

LEDAcrypt provides a KEM, named LEDAcrypt KEM, employing the Niederreiter cryptosystem with One Wayness against Chosen Plaintext Attack (OW-CPA) and an apt conversion to obtain a KEM with a twofold goal: (i) provide a fast KEM with indistinguishability under chosen plaintext attack (IND-CPA) and ephemeral keys for low latency session establishment with perfect forward secrecy, and (ii) provide a KEM with IND-CCA2 and long term keys for scenarios where key reuse may be desirable. We achieve this goal employing the same IND-CCA2 conversion applied to the QC-LDPC Niederreiter cryptosystem for both scenarios and achieving an appropriate DFR through code parameter tuning and, where needed, an additional IND-CCA2 redundant encryption technique. In particular, we employ the $U^{\not\perp}_{m}$ construction defined in [18], which starts from a deterministic cryptosystem to build a KEM with IND-CCA2 in the Random Oracle Model (ROM) with a tight reduction.

The same construction was proven to achieve IND-CCA2 in the Quantum Random Oracle Model (QROM) in [19], with a tighter security reduction being reported in [20], starting from the assumption that the underlying deterministic cryptosystem is OW-CPA, as it is the case with our Niederreiter KEM. The proofs in [18,19,20] take into account the possibility that the underlying cryptoscheme is characterized by a bounded correctness error δ. The instantiation of the $U^{\not\perp}_{m}$ construction employing the QC-LDPC code-based Niederreiter cryptoscheme and a cryptographically secure hash, $\texttt{Hash}(\cdot)$, is reported in Fig. 1. We chose, as the cryptographically secure hash to instantiate the $U^{\not\perp}_{m}$ construction, the NIST standard SHA-3 hash function.

In case of a decoding failure [18,19,20], the decapsulation procedure computes the returned outcome by hashing a secret value and the ciphertext. This prevents an adversary from distinguishing when a failure occurs due to malformed plaintext messages, i.e., messages with a number of asserted bits that is not exactly equal to t, from when a failure occurs due to the intrinsic behavior of the underlying QC-LDPC code. In other terms, the adversary cannot draw any conclusion about the decoding abilities of the code at hand when he/she is in control of composing messages that are not in the legitimate message space.

To provide IND-CCA2 for a given security level 2^{λ}, the authors of [18] state that it is required for the decryption function to have a correctness error $\delta \leq 2^{-\lambda}$. Given our goal of having both a fast and compact KEM with ephemeral keys and IND-CPA guarantees, as well as a KEM with IND-CCA2, we will provide different sets of parameters to be employed in the LEDAcrypt KEM construction,

Algorithm 1: LEDAcrypt-KEM ENCAP

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

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

(a)

Algorithm 2: LEDAcrypt-KEM DECAPS

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

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

(b)

Fig. 1. Key encapsulation (a) and key decapsulation (b) primitives of LEDAcrypt KEM

which are characterized by a DFR low enough to foil statistical attacks and achieve IND-CCA2 guarantees, without hindering practical deployment.

In addition to the LEDAcrypt KEM, LEDAcrypt provides a public-key cryptosystem (PKC) with IND-CCA2 guarantees. While it is possible to employ LEDAcrypt KEM in a Key Encapsulation Module + Data Encapsulation Mechanism (KEM+DEM) combination with a symmetric encryption primitive, we note that such an approach may lead to a non-negligible ciphertext expansion in case plaintexts are small in size. To overcome such an issue, LEDAcrypt PKC provides a construction that starts from the QC-LDPC code-based McEliece cryptosystem and derives a PKC exploiting the available capacity of the McEliece

Public-Key Encryption (PKE) primitive to store the actual message content. It is worth noting that, in the McEliece setting, the systematic form of the generator matrix of the public code included in the public key would easily allow any observer to recover the information word embedded in an encrypted message, without recovering the private key. Nevertheless, the conversion proposed by Kobara and Imai in [23], with the purpose of maximizing the amount of message encrypted by a McEliece PKC, allows IND-CCA2 guarantees to be provided in the ROM. Therefore, the confidentiality of the information word as well as the security of the private key remain guaranteed by the hardness of the NP-hard general decoding problem even when a systematic generator matrix is employed as public key. For a detailed description of the basic encryption and decryption transformations of LEDAcrypt PKC, as well as the mechanisms of the γ-conversion scheme [23] that allow us to obtain an IND-CCA2 version of LEDAcrypt PKC, we refer the reader to the LEDAcrypt specification [3].

3 Security level goals

The bar to be cleared to design parameters for post-quantum cryptosystems was set by NIST to the computational effort required on either a classical or a quantum computer to break the AES with a key size of λ bits, $\lambda \in \{128, 192, 256\}$, through an exhaustive key search. The three pairs of computational efforts required on a classical and quantum computer correspond to NIST Category 1, 3, and 5, respectively [31]. Throughout the design of the parameters for LEDAcrypt we ignore Categories 2 and 4: if a cipher matching those security levels is required, we advise to employ the parameters for Categories 3 and 5, respectively.

The computational worst-case complexity of breaking AES on a classical computer can be estimated as $2^{\lambda}\mathtt{C}_{\mathtt{AES}}$, where $\mathtt{C}_{\mathtt{AES}}$ is the amount of binary operations required to compute AES on a classical computer on a small set of plaintexts, and match them with a small set of corresponding ciphertexts to validate the correct key retrieval. Indeed, more than a single plaintext-ciphertext pair is required to retrieve AES keys [14]. In particular, a validation on three plaintext-ciphertext pairs should be performed for AES-128, on four pairs for AES-192 and on five for AES-256.

Willing to consider a realistic AES implementation for exhaustive key search purposes, we refer to [38], where the authors survey the state of the art of Application-Specific Integrated Circuit (ASIC) AES implementations, employing the throughput per Gate Equivalent (GE) as their figure of merit. The most performing AES implementations are the ones proposed in [38], and require around 16ki GEs. We thus deem reasonable to estimate the computational complexity of an execution of AES as 16ki binary operations. We are aware of the fact that this is still a conservative estimate, as we ignore the cost of the interconnections required to carry the required data to the AES cores.

The computational complexity of performing an AES key retrieval employing a quantum computer was measured first in [14], where a detailed implementation of an AES breaker is provided. The computation considers an implementation

Table 1. Classical and quantum computational costs of a key search on AES

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

of Grover's algorithm [15] seeking the zeros of the function given by the binary comparison of a set of AES ciphertexts with the encryption of their corresponding plaintexts for all the possible key values. The authors of [14] chose to report the complexity of the quantum circuit computing AES counting only the number of the strictly needed Clifford and T gates, since they are the ones currently most expensive to implement in practice. Selecting a different choice for the set of quantum gates employed to realize the AES circuit may yield a different complexity; however, the difference will amount to a reasonably small constant factor, as it is possible to re-implement the Clifford and T gates at a constant cost with any computationally complete set of quantum gates. We thus consider the figures reported in [14] as a reference for our parameter design procedure. In Table 1 we summarize the computational cost of performing exhaustive key searches on all three AES variants (i.e., with 128, 192, and 256 bits long keys), both considering classical and quantum computers.

4 Evaluated attacks

Let us briefly recall the set of attacks to be considered in the design of the system parameters. In addition to those advanced attacks, we also consider some basic attack procedures, such as exhaustive key search, which must be taken into account in any automated cryptosystem parameter optimization, since they impose some bounds on the system parameters.

An open source software implementation of the routines for computing the complexity of the described attacks is publicly available [1].

4.1 Attacks based on exhaustive key search

Enumerating all the possible values for the secret key is, in principle, applicable to any cryptosystem. The original LEDAkem and LEDApkc specification documents do not mention exhaustive key search, as it is possible to verify that they are strictly dominated by other, less computationally demanding, attack strategies such as the use of ISD algorithms.

In this parameter revision, in order to pose suitable bounds to the automated parameter search we perform, we consider the application of an exhaustive enumeration strategy to each one of the two secret low-density binary matrices

constituting the LEDAcrypt secret keys, i.e., H and Q. We recall that H is a block circulant binary matrix constituted by $1 \times n_0$ circulant blocks with size equal to p bits, where $n_0 \in \{2, 3, 4\}$ and p is a prime such that $\mathrm{ord}_2(p) = p - 1$ (i.e., $2^{p-1} \bmod p = 1 \bmod p$). Q is a binary block circulant matrix constituted by $n_0 \times n_0$ binary circulant blocks with size p. Willing to follow a conservative approach, we design revised parameter sets such that it is not possible for an attacker to enumerate all the possible matrices H or Q. While there is no standing attack benefiting from such an enumeration, we deem reasonable adding such a constraint to the design of the parameter sets as a peace-of-mind measure. In our approach, to prevent attacks relying on the exhaustive search for the value of either H or Q, we considered the remainder of the attack strategy which may be employed to derive the matrix not being exhaustively searched for to have a constant complexity (i.e. $\Theta(1)$). This in turn implies that any attack strategy which leverages the exhaustive search of H or Q to obtain useful information for a key recovery attack will have a computational complexity matching or exceeding the required security level.

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

Considering the case where the key recovery strategy exploits the enumeration of either H or Q within an algorithm running on a quantum computer, we consider the possibility of employing a Grover-like strategy to speedup the enumeration of either H or Q. Assuming conservatively that such a strategy exists, we consider the resistance against exhaustive key search with a quantum computer to be $\sqrt{\sharp H}$ and $\sqrt{\sharp Q}$ for the search over H and Q, respectively. We note that, for all parameter sets proposed in the original specification [3], the cost of enumerating H and Q exceeds that of the best attacks via ISD.

4.2 Attacks based on information set decoding

It is well known that efficient message and key recovery attacks against McEliece and Niederreiter cryptosystem variants based on low-density (LDPC) and moderate-density (MDPC) parity-check codes are those exploiting ISD algorithms. Such algorithms have a long development history, dating back to the early '60s [34], and provide a way to recover the error pattern affecting a codeword of a generic random linear block code given a representation of the code in the form of either its generator or parity-check matrix.

Despite the fact that the improvement provided by ISD over the straightforward enumeration of all the possible error vectors affecting the codeword is only polynomial, employing ISD provides substantial speedups. It is customary for

ISD variant proposers to evaluate the effectiveness of their attacks considering the improvement on a worst-case scenario as far as the code rate and number of corrected errors goes (see, for instance [5]). Such an approach allows deriving the computational complexity as a function of a single variable, typically taken to be the code length n, and obtaining asymptotic bounds for the behavior of the algorithms. In our parameter design, however, we chose to employ non-asymptotic estimates of the computational complexity of the ISD attacks. Therefore, we explicitly compute the amount of time employing a non-asymptotic analysis of the complexity of ISD algorithms, given the candidate parameters of the code at hand. This approach also permits us to retain the freedom to pick rates for our codes which are different from the worst-case one for decoding, thus exploring different trade-offs in the choice of the system parameters. In case the ISD algorithm has free parameters, we seek the optimal case by explicitly computing the complexity for a large region of the parameter space, where the minimum complexity resides. We consider the ISD variants proposed by Prange [34], Lee and Brickell [24], Leon [25], Stern [36], Finiasz and Sendrier [11], and Becker, Joux, May and Meurer (BJMM) [5], in our computational complexity evaluation on classical computers. The reason for considering all of them is to avoid concerns on whether their computational complexity in the finite-length regime is already well approximated by their asymptotic behavior. In order to estimate the computational complexity of ISD on quantum computing machines, we consider the results reported in [8], which are the same employed in the original specification [3]. Since complete and detailed formulas are available only for the ISD algorithms proposed by Lee and Brickell, and Stern [36], we consider those as our computational complexity bound. While asymptotic bounds show that executing a quantum ISD derived from the May-Meurer-Thomae (MMT) algorithm [26] is faster than a quantum version of Stern's [21], we note that there is no computational complexity formulas available for generic code and error rates.

Message recovery attacks through ISD. ISD algorithms can effectively be applied to recover the plaintext message of any McEliece or Niederreiter cryptosystem instance by retrieving the intentional error pattern used during encryption. When a message recovery attack of this kind is performed against a system variant exploiting quasi cyclic codes, like those at hand, it is known that a speedup equal to the square root of the circulant block size can be achieved [35]. We consider such message recovery attacks in our parameter design, taking this speedup into account in our computations.

Key recovery attacks through ISD. The most efficient way, and currently the only known way, to exploit the sparsity of the parity checks that characterizes the secret code $H' = HQ$ in order to attack LEDAcrypt is trying to recover a low-weight codeword of the dual of the public code. In fact, any sparse row of H' is a low-weight codeword belonging to the dual of the public code, and such codewords have a weight that is very close or equal to $d' = n_0 d_v(\sum_{i=0}^{n_0-1} m_i)$, which is comparatively small with respect to the codeword length n.

Therefore, it is possible to search for such low-weight codewords through ISD algorithms, which are far more efficient than trying all the $\binom{n}{d'}$ possible codewords. Indeed, the complexity of accomplishing this task through ISD is equal to the one of decoding a code of the same length n, with dimension equal to the redundancy $r = n - k$ of the code at hand, and with d' errors.

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

4.3 Reaction attacks

In addition to the proper sizing of the parameters of LEDAcrypt so that it withstands the aforementioned attacks, a last concern should be taken into account regarding the lifetime of a LEDAcrypt key pair, when keys are not ephemeral. In fact, whenever an attacker may gain access to a decryption oracle to which he may pose a large amount of queries, the so-called *reaction attack* becomes applicable. Reaction attacks recover the secret key by exploiting the inherent non-zero DFR of QC-LDPC codes [9,10,17]. In particular, these attacks exploit the correlation between the DFR of the code, the positions of the parity checks in the private matrix, and the erroneous positions in the error vector. Indeed, whenever e and either H' have pairs of ones placed at the same distances, the decoder exhibits a DFR smaller than the average.

Such attacks require the collection of the outcome of decoding (success or failure) on a ciphertext for which the attacker knows the distances in the support of the error vector, for a significant number of ciphertexts, to achieve statistical confidence in the result. The information on the decoding status is commonly referred to as the *reaction* of the decoder, hence the name of the attack. The strongest countermeasure against these attacks is to choose a proper set of system parameters such that the DFR is negligible, which means an attacker would require a computationally unfeasible amount of decryption actions to obtain even a single decoding failure. It has been recently pointed out in [33] that some mechanisms exist to generate error vectors able to artificially increase the DFR of systems such as LEDAcrypt. However, such techniques require to known an error vector causing a single decoding failure to be carried out. Therefore, choosing an appropriately low DFR such attacks can be made as expensive as a key recovery via ISD. In addition, these methods require the manipulation of error vectors, which is not feasible when an IND-CCA2 secure conversion is adopted.

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

Reaction attacks could instead be attempted in the case of an accidental reuse of the keys in these instances. However, the parameter choices of LEDAcrypt

KEM instances with ephemeral keys guarantee a DFR in the order of 10^{-8}–10^{-9}. An attacker would need to collect DFR^{-1} ciphertexts encrypted with the same key, on average, before observing one decryption failure. Hence, these instances are protected against a significant amount of accidental key reuse. Moreover such a low DFR provides a good practical reliability for the ephemeral KEM, as it is very seldom needed to repeat a key agreement due to a decoding failure.

Instances with long term keys LEDAcrypt KEM and LEDAcrypt PKC instances with long term keys employ a suitable conversion to achieve IND-CCA2. The IND-CCA2 model assumes that an attacker is able to create a polynomially bound number of chosen ciphertexts and ask for their decryption to an oracle owning the private key corresponding to the public key used for their generation. Note that such an attacker model includes reaction attacks, since the adversary is able to observe a large number of decryptions related to the same keypair.

A noteworthy point is that the current existing IND-CCA2 constructions require the DFR of the scheme to be negligible. Indeed, most IND-CCA2 attaining constructions require the underlying cryptosystem to be correct, i.e., $D_{sk}(E_{pk}(m)) = m$, for all valid key pairs (pk, sk) and for all valid messages m. Recent works [20,19] tackled the issue of proving a construction IND-CCA2 even in the case of an underlying cipher affected by decryption failures. The results obtained show that, in case the DFR is negligible in the security parameter, it is possible for the construction to attain IND-CCA2 guarantees even in case of decryption failures.

In systems with non-zero DFR, endowed with IND-CCA2 guarantees, attacks such as crafting a ciphertext aimed at inducing decoding errors are warded off. Therefore, our choice of employing an IND-CCA2 achieving construction to build both our PKC and KEM, paired with appropriate parameters guaranteeing a negligible DFR allows us to thwart ciphertext alteration attacks such as the ones pointed out in the official comments to the first round of the NIST competition[3].

5 Parameter design

In this section we describe an automated procedure for the design of parameters for the QC-LDPC codes employed in LEDAcrypt. An open source software implementation of the routines for computing the complexity of the described attacks and perform parameter generation is available as public domain software at [1]. The LEDAcrypt design procedure described in this section takes as input the desired security level λ_c and λ_q, expressed as the base-2 logarithm of the number of operations of the desired computational effort on a classical and quantum computer, respectively. In addition to λ_c and λ_q, the procedure also takes as input the number of circulant blocks, $n_0 \in \{2, 3, 4\}$, forming the parity-check matrix H, allowing tuning of the code rate. As a third and last parameter, the

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

procedure takes as input the value of ϵ, which tunes the estimate of the system DFR for the IND-CPA case. We first consider instances with ephemeral keys, which are designed using $\epsilon = 0.3$: the resulting DFR values are in the range 10^{-9}–10^{-8}. The design of parameters for long term keys starts from the output of the procedure employed for ephemeral key parameters, increasing the value of p until the bounds specified in Section 7 provide a sufficiently small DFR.

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

- Classical and quantum exhaustive searches for the values of H or Q should require at least 2^{λ_c} and 2^{λ_q} operations. This constraint binds the value of the circulant block size p and the weight of a row of the circulant block, d_v for H and m_i for Q, to be large enough.
- The minimum cost for a message recovery via ISD on both quantum and classical computers must exceed 2^{λ_q} and 2^{λ_c} operations, respectively. This constraint binds the values of the code length $n = n_0 p$, the code dimension $k = (n_0 - 1)p$ and the number of errors t to be chosen such that an ISD on the code $\mathcal{C}(n, k, t)$ requires more than 2^{λ_q} or 2^{λ_c} operations on a quantum and a classical computer.
- The minimum cost for a key recovery attack via ISD on both quantum and classical computers must exceed 2^{λ_q} and 2^{λ_c} operations, respectively. This constraint binds the values of the code length $n = n_0 p$, the code redundancy $r = p$ and the number of ones in a row of HQ, $d'_v n_0$, with $d'_v = d_v m$ to be chosen such that an ISD on the code $\mathcal{C}(n, r, d'_v n_0)$ requires more than 2^{λ_q} or 2^{λ_c} operations on a quantum and classical computer.
- The choice of the circulant block size, p, should be such that p is a prime number and $\mathrm{ord}_2(p) = p - 1$ in order to ensure non-singularity of the last block of H'.
- The choice of the circulant block size, p, and parity-check matrix density, $n_0 d_v$, must allow the code to correct the required amount of errors. This is tested through the computation of the decoding threshold, as described in the original specification [3].
- The weights of the circulant blocks of Q must be such that the permanent of the matrix of the block weights is odd, which guarantees the existence of its multiplicative inverse (see the full LEDAcrypt specification [3] for details).

We report a synthetic description of the procedure implemented in the publicly available code as Algorithm 3. The rationale of the procedure[4] is to proceed in refining the choice for p, t, d_v, and all the m_i's at fix point, considering only values of p respecting $\mathrm{ord}_2(p) = p - 1$.

[4] Note that, in the pseudocode of Algorithm 3, the loop construct **while**(< condition >) ... iterates the execution of instructions in the loop body when the condition is **true**, while the loop construct **Repeat** ... **until**(< condition >) iterates the instructions in the loop body when the condition is **false**.

Algorithm 3: LEDAcrypt Parameter Generation

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

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

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

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

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

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

To determine the weight of a column of H, i.e., d_v and the weight of a column of Q, i.e., m, with $m = \sum_{i=0}^{n_0-1} m_i$, the procedure moves on searching for a candidate value of d'_v, where $d'_v = d_v m$ and $d'_v n_0$ is the weight of a row of HQ. Given a value for d'_v (line 8 and line 21), the value of d_v is computed as the smallest odd integer greater than the square root of d'_v (line 10). The condition of d_v being odd is sufficient to guarantee the non singularity of the circulant blocks of H, while the square root computation is meant to distribute the weight d'_v evenly between the weight of a column of H and the weight of a column of Q. The weight of a column of Q, i.e., m, is then computed through the loop in lines 11–15. Specifically, the value of m must allow a partition into n_0 integers (i.e., $m = \sum_{i=0}^{n_0-1} m_i$) such that the permanent of the circulant integer matrix having the said partition as a row is odd, for the matrix Q to be invertible [3]. Therefore, in the loop body the value of m is assumed as $\left\lceil \frac{d'_v}{d_v} \right\rceil$ (line 13) and subsequently checked to derive the mentioned partition in n_0 integers. The loop (lines 11–15) ends when either a valid partition of m is found or m turns out to be smaller than the number of blocks n_0 (as finding a partition in this case would be not possible increasing only the value of d_v).

Algorithm 3 proceeds to test for the security of the cryptosystem against key recovery attacks and key enumeration attacks on both classical and quantum computers (lines 16–18). If a legitimate value for m has not been found, the current parameters of the cryptoystem are deemed insecure (line 20). In line 21, the current value of d'_v is incremented by one and another iteration of the loop is executed if the security constraints are not met with the current parameters (i.e., `SecureOk` = **false**) and it is still viable to perform another iteration to check the updated value of d'_v, i.e., $d'_v n_0 < p$ (line 22).

If suitable values for the code parameters from a security standpoint are found, the algorithm computes the minimum value of p, named p_{th}, such that the decoding algorithm is expected to correct t errors, according to the methodology reported in [3] (see lines 23–24); otherwise, the value of p_{th} is forced to be equal to p (lines 25–26) in such a way that another iteration of the outer loop of Algorithm 3 is executed through picking a larger value of p and new values for the remaining parameters.

We note that, while the decoding threshold provides a sensible estimate of the fact that the QC-LDPC code employing the generated parameters will correct the computed amount of errors, this is no substitute for a practical DFR evaluation, which is then performed through Monte Carlo simulations. Willing

Table 2. Parameter sizes for LEDAcrypt KEM obtained with the parameter design tool, compared to those of LEDAkem appearing in the original specification

NIST Cat.	n_0	LEDAcrypt KEM p	t	d_v	m	errors out of decodes	LEDAkem original p	t	d_v	m	errors out of decodes
1	2	14,939	136	11	[4, 3]	14 out of $1.2 \cdot 10^9$	27,779	224	17	[4, 3]	19 out of $2.22 \cdot 10^9$
	3	7,853	86	9	[4, 3, 2]	0 out of $1 \cdot 10^9$	18,701	141	19	[3, 2, 2]	0 out of $1 \cdot 10^9$
	4	7,547	69	13	[2, 2, 2, 1]	0 out of $1 \cdot 10^9$	17,027	112	21	[4, 1, 1, 1]	0 out of $1 \cdot 10^9$
3	2	25,693	199	13	[5, 3]	2 out of $1 \cdot 10^9$	57,557	349	17	[6, 5]	0 out of $1 \cdot 10^8$
	3	16,067	127	11	[4, 4, 3]	0 out of $1 \cdot 10^9$	41,507	220	19	[3, 4, 4]	0 out of $1 \cdot 10^8$
	4	14,341	101	15	[3, 2, 2, 2]	0 out of $1 \cdot 10^9$	35,027	175	17	[4, 3, 3, 3]	0 out of $1 \cdot 10^8$
5	2	36,877	267	11	[7, 6]	0 out of $1 \cdot 10^9$	99,053	474	19	[7, 6]	0 out of $1 \cdot 10^8$
	3	27,437	169	15	[4, 4, 3]	0 out of $1 \cdot 10^9$	72,019	301	19	[7, 4, 4]	0 out of $1 \cdot 10^8$
	4	22,691	134	13	[4, 3, 3, 3]	0 out of $1 \cdot 10^9$	60,509	239	23	[4, 3, 3, 3]	0 out of $1 \cdot 10^8$

to target a DFR of 10^{-9}, we enlarged heuristically the value of p until the target DFR was reached (adding 5% of the value of p). Enlargements took place for:

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

The C++ tool[5] provided follows the computation logic described in Algorithm 3, but it is optimized to reduce the computational effort as follows:

- The search for the values of t and d'_v respecting the constraints is performed by means of a dichotomic search instead of a linear scan of the range.
- The computations of the binomial coefficients employ a tunable memorized table to avoid repeated re-computation, plus a switch to Stirling's approximation (considering the approximation up to the fourth term of the series) only in the case where the value of $\binom{a}{b}$ is not available in the table and $b > 9$. In case the value of the binomial is not available in the table and $b < 9$ the result is computed with the iterative formula for the binomial, to avoid the discrepancies between Stirling's approximation and the actual value for small values of b.
- The values of p respecting the constraint $\mathrm{ord}_2(p) = p - 1$ are pre-computed up to 159,979 and stored in a lookup table.
- The search for the value of p is not performed scanning linearly the aforementioned table. The strategy to find the desired p starts by setting the value of the candidate for the next iteration to $\textsc{NextPrime}(\lceil (1+\epsilon)p_{th} \rceil)$ up to finding a value of p, $\bar{p}$ which satisfies the constraints. Subsequently

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

Table 3. Computational cost of an exhaustive enumeration attack on either the matrix H or the matrix Q. The quantum execution model considers the possibility of attaining the full speedup yielded by the application of Grover's algorithm to the computation

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

the algorithm starts scanning the list of primes linearly from $\bar{p}$ backwards to find the smallest prime which satisfies the constraints.

6 LEDAcrypt instances with ephemeral keys

In Table 2 we provide parameters for LEDAcrypt KEM instances employing ephemeral keys, and compare them with those of LEDAkem appearing in the original specification. This shows how the new parameterization is tighter and enables a considerable reduction in the key sizes. Deriving them took approximately a day for all the parameter sets with $n_0 \in \{3, 4\}$ and approximately a month for all the parameter sets with $n_0 = 2$ on a dual socket AMD EPYC 7551 32-Core CPU. The memory footprint for each parameter seeking process was below 100 MiB.

6.1 Resulting computational complexity of attacks

When an algorithmic procedure is exploited for the design of parameter sets, as in our case, some constraints on the choice of the row/column weights of H and Q must be imposed in such a way as to make enumeration of either H or Q unfeasible to an attacker. Therefore, enumeration attacks of the type described in Section 4.1 must be taken into account. In Table 3 we report the computational cost of performing such an exhaustive enumeration, both with a classical and a quantum computer. The latter has been obtained by applying the speedup due to Grover's algorithm to the complexity computed considering a classical computer. From the results in Table 3 it is straightforward to note that an exhaustive search on either H or Q is above the required computational effort.

Table 4. Cost of performing a message recovery attack, i.e., an ISD on the code $\mathcal{C}(n_0 p, (n_0 - 1)p, t)$, for LEDAcrypt instances with the parameters p, t reported in Table 2, employing the considered ISD variants.

NIST Cat.	n_0	Classical computer ($\log_2$ ♯binary op.s)						Quantum computer ($\log_2$ ♯quantum gates)	
		Prange [34]	L-B [24]	Leon [25]	Stern [36]	F-S [11]	BJMM [5]	Q-LB [8]	Q-Stern [8]
1	2	169.05	158.23	156.35	148.59	148.57	144.37	97.26	98.67
	3	167.72	157.37	154.51	147.67	147.65	144.29	96.14	97.55
	4	169.62	159.40	155.86	149.32	149.31	145.98	97.47	98.22
3	2	234.11	222.19	220.26	210.42	210.41	207.17	130.22	131.62
	3	235.32	223.84	220.82	211.91	211.90	208.71	130.67	132.07
	4	235.98	224.66	220.97	212.39	212.39	209.10	131.26	132.66
5	2	303.56	290.79	288.84	277.40	277.39	274.54	165.18	166.58
	3	303.84	291.53	288.42	277.98	277.98	274.34	165.48	166.88
	4	303.68	291.54	287.78	277.67	277.67	274.91	165.52	166.92

As described in Section 4.2, the two main attacks that can be mounted against the considered systems are message recovery attacks and key recovery attacks based on ISD algorithms. Table 4 and Table 5 report the complexities of these attacks against LEDAcrypt instances employing the parameters p, t in Table 2. An interesting point to be noted is that, while providing clear asymptotic speedups, the improvements to the ISD algorithms proposed since Stern's [36] are only able to achieve a speedup between 2^2 and 2^4 when their finite regime complexities are considered in the range of values concerning LEDAcrypt cryptosystem parameters. Concerning quantum ISDs, it is interesting to notice that the quantum variant of the Stern's algorithm as described by de Vries [8] does not achieve an effective speedup when compared against a quantum transposition of Lee and Brickell's ISD. Such a result can be ascribed to the fact that the reduction in the number of ISD iterations which can be obtained by Stern's ISD is mitigated by the fact that the applying Grover's algorithm to the iterations themselves cuts their number (and Stern's reduction factor) quadratically [7].
Comparing the computational complexities of the message recovery attack (Table 4) and the key recovery attack (Table 5), we note that performing a message recovery attack is almost always easier than the corresponding key recovery attack on the same parameter set, albeit by a small margin.

7 LEDAcrypt instances with long term keys

For LEDAcrypt instances employing long term keys, we need that the DFR is sufficiently small to enable IND-CCA2. However, such small values of DFR cannot be assessed through Monte Carlo simulations. Hence, for these instances we consider a Q-decoder performing two iterations and exploit the analysis reported in Appendix A in order to characterize its DFR. To this end, we consider a two-stage rejection sampling during key generation, that is:

Table 5. Cost of performing a key recovery attack, i.e., an ISD on the code $\mathcal{C}(n_0 p, p, n_0 d_v m)$, for the revised values of the parameters p, n_0, d_v, m reported in Table 2, employing the considered ISD variants

NIST Cat.	n_0	Classical computer ($\log_2$ ♯binary op.s)						Quantum computer ($\log_2$ ♯quantum gates)	
		Prange [34]	L-B [24]	Leon [25]	Stern [36]	F-S [11]	BJMM [5]	Q-LB [8]	Q-Stern [8]
1	2	180.25	169.06	167.24	158.76	158.75	154.94	99.21	100.62
	3	169.36	157.78	156.53	147.71	147.68	144.08	93.93	95.34
	4	179.86	167.79	165.69	157.13	157.10	153.01	99.71	101.12
3	2	237.85	225.77	223.87	213.72	213.71	210.64	128.35	129.75
	3	241.70	228.98	227.03	216.59	216.57	213.18	130.56	131.96
	4	254.92	241.73	238.97	228.80	228.78	224.76	137.60	139.01
5	2	315.08	302.11	300.19	288.35	288.34	285.71	167.04	168.44
	3	320.55	306.93	304.48	292.78	292.77	289.00	170.31	171.71
	4	312.68	298.84	295.66	284.59	284.58	280.91	166.82	168.22

1. Only pairs of H and Q such that the weight of a column of $H' = HQ$ is $d_v m$.
2. Only pairs of H and Q are retained for which the condition (2) in Appendix A.1 guaranteeing low enough DFR is verified.

The first rejection sampling is useful to achieve a constant computational effort for message and key recovery attacks. In addition, it simplifies the second rejection sampling, which can take advantage of a special case of the analysis reported in Appendix A. Then, the second rejection sampling is performed to ensure that the generated key pair can achieve with two iterations of the Q-decoder the correction of all residual errors of weight $\leq \bar{t}$ left by the first decoder iteration. Such a property can be obtained with the knowledge of the generated matrices H and Q alone. If the above condition is not satisfied the generated key pair is discarded, and the procedure is repeated until a valid key pair is found. For valid key pairs, achieving a desired target DFR value, $\overline{\text{DFR}}$, is guaranteed by the choice of code parameters such that the first iteration of the Q-decoder results in at most $\bar{t}$ residual errors with probability $> 1 - \overline{\text{DFR}}$.

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

Some choices are reported in Table 6, by imposing a DFR bounded by either 2^{-64} or $2^{-\lambda}$, where λ equals 128, 192, 256 for NIST Category 1, 3, 5, respectively. The proposed choices aim at parameter sets for which the probability of drawing a random secret key achieving the DFR target is significant to minimize key generation overhead. To design these parameters, we start from the ones obtained through the automated parameter optimization procedure used for instances with ephemeral keys previously described, while keeping the product md_v constant or slightly increased. Then, we proceed by increasing the size

Table 6. Parameters for the LEDAcrypt KEM with long term keys and the LEDAcrypt PKC employing a two-iteration Q-decoder matching a DFR equal to 2^{-64} and a DFR equal to $2^{-\lambda}$, where λ equals 128, 192, 256 for NIST Category 1, 3, 5, respectively

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

of the circulant blocks, until we obtain a probability smaller than the given target that the number of bit errors that are left uncorrected by the first iteration is $\leq \bar{t}$. In particular, such a probability is computed by considering all possible choices for the flipping threshold of the first iteration, and by taking the optimal one (i.e., the one corresponding to the maximum value of the probability). Note that such changes may only impact positively on the security margin against ISD attacks and key enumeration attacks.

For any set of parameters so designed, we draw 100 key pairs at random, and evaluate how many of them satisfy the condition (2) in Appendix A. As it can be seen from the results reported in Table 6, the parameter sets we determined are able to achieve a DFR $< 2^{-64}$ increasing the key size by a factor ranging from $2\times$ to $3\times$ with respect to the case of ephemeral key pairs.

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

8 Conclusion

In this work we presented a threefold contribution on code-based cryptosystems relying on QC-LDPC codes, such as the NIST post-quantum candidate LEDAcrypt. First of all, we quantify the computational effort required to break AES both via classical and quantum computation, providing a computational effort to be matched in post-quantum cryptosystem parameter design. Our second contribution is an automated optimization procedure for the parameter generation of ephemeral-key cryptosystem parameters for LEDAcrypt, providing a 10^{-9}–10^{-8} DFR, low enough for practical use. Our third contribution is a closed form characterization of the decoding strategy employed in the LEDAcrypt systems that allows to obtain an upper bound on their DFR at design time, in turn

the generation of cryptosystem parameter sets which guarantee a bounded DFR. This in turn allows to generate cryptosystem parameters achieving IND-CCA2 guarantees, which ensure that active attacks, including reaction attacks, have a computational cost exceeding the desired security margin, without the need for ephemeral keys. We note that the proposed bound on the DFR can be fruitfully integrated in the automated design procedure proposed in this paper.

References

1. LEDAtools. https://github.com/LEDAcrypt/LEDAtools, 2019.
2. M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini. LEDAkem: A post-quantum key encapsulation mechanism based on QC-LDPC codes. In T. Lange and R. Steinwandt, editors, *Post-Quantum Cryptography*, LNCS, pages 3–24. Springer International Publishing, Cham, 2018.
3. M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini. LEDAcrypt website. https://www.ledacrypt.org/, 2019.
4. 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*, volume 5229 of *LNCS*, pages 246–262. Springer Verlag, 2008.
5. 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 D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012.
6. E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Trans. Inform. Theory*, 24(3):384–386, May 1978.
7. D. J. Bernstein. Grover vs. mceliece. In N. Sendrier, editor, *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings*, volume 6061 of *Lecture Notes in Computer Science*, pages 73–80. Springer, 2010.
8. S. de Vries. Achieving 128-bit Security against Quantum Attacks in OpenVPN. Master's thesis, University of Twente, August 2016.
9. T. Fabšič, V. Hromada, P. Stankovski, P. Zajac, Q. Guo, and T. Johansson. A reaction attack on the QC-LDPC McEliece cryptosystem. In T. Lange and T. Takagi, editors, *Post-Quantum Cryptography: 8th International Workshop, PQCrypto 2017*, pages 51–68. Springer, Utrecht, The Netherlands, June 2017.
10. T. Fabsic, V. Hromada, and P. Zajac. A reaction attack on LEDApkc. *IACR Cryptology ePrint Archive*, 2018:140, 2018.
11. M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009.
12. P. Gaborit. Shorter keys for code based cryptography. In *Proc. Int. Workshop on Coding and Cryptography (WCC 2005)*, pages 81–90, Bergen, Norway, Mar. 2005.
13. R. G. Gallager. *Low-Density Parity-Check Codes*. M.I.T. Press, 1963.

14. M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt. Applying Grover's Algorithm to AES: Quantum Resource Estimates. In T. Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2016.
15. L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. 28th Annual ACM Symposium on the Theory of Computing*, pages 212–219, Philadephia, PA, May 1996.
16. Q. Guo, T. Johansson, and P. Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 789–815. Springer Berlin Heidelberg, 2016.
17. Q. Guo, T. Johansson, and P. Stankovski Wagner. A key recovery reaction attack on QC-MDPC. *IEEE Trans. Information Theory*, 65(3):1845–1861, Mar. 2019.
18. D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Y. Kalai and L. Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, 2017.
19. H. Jiang, Z. Zhang, L. Chen, H. Wang, and Z. Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 96–125. Springer, 2018.
20. H. Jiang, Z. Zhang, and Z. Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. Cryptology ePrint Archive, Report 2019/134, to appear in PQCrypto 2019, 2019. https://eprint.iacr.org/2019/134.
21. G. Kachigar and J. Tillich. Quantum information set decoding algorithms. In T. Lange and T. Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2017.
22. R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.
23. K. Kobara and H. Imai. Semantically secure McEliece public-key cryptosystems — conversions for McEliece PKC. *Lecture Notes in Computer Science*, 1992:19–35, 2001.
24. P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988.
25. J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. Information Theory*, 34(5):1354–1359, 1988.

26. A. May, A. Meurer, and E. Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In D. H. Lee and X. Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011.
27. R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report*, pages 114–116, 1978.
28. R. Misoczki and P. S. L. M. Barreto. Compact McEliece keys from Goppa codes. In *Selected Areas in Cryptography*, volume 5867 of *LNCS*, pages 376–392. Springer, 2009.
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 *Proc. IEEE International Symposium on Information Theory (ISIT 2000)*, pages 2069–2073, July 2013.
30. C. Monico, J. Rosenthal, and A. Shokrollahi. Using low density parity check codes in the McEliece cryptosystem. In *Proc. IEEE International Symposium on Information Theory (ISIT 2000)*, page 215, Sorrento, Italy, June 2000.
31. National Institute of Standards and Technology. Post-quantum crypto project, Dec. 2016.
32. H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Probl. Contr. and Inf. Theory*, 15:159–166, 1986.
33. A. Nilsson, T. Johansson, and P. Stankovski Wagner. Error amplification in code-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):238–258, nov 2018.
34. E. Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Information Theory*, 8(5):5–9, 1962.
35. N. Sendrier. Decoding one out of many. In B. Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, volume 7071 of *Lecture Notes in Computer Science*, pages 51–67. Springer, 2011.
36. J. Stern. A method for finding codewords of small weight. In G. D. Cohen and J. Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.
37. J. Tillich. The decoding failure probability of MDPC codes. In *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*, pages 941–945, 2018.
38. R. Ueno, S. Morioka, N. Homma, and T. Aoki. A High Throughput/Gate AES Hardware Architecture by Compressing Encryption and Decryption Datapaths - Toward Efficient CBC-Mode Implementation. In B. Gierlichs and A. Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 538–558. Springer, 2016.

A Bounded DFR for Q-decoders

Binary block error correction codes $\mathcal{C}(n, k, t)$ with a low density $r \times n$ parity check matrix H' allow iterative decoding strategies which aim at solving at fix

point the simultaneous binary equation system given by $s = H'e^T$, where $s \in \mathbb{Z}_2^r$ is a $1 \times r$ binary vector named *syndrome*, $e \in \mathbb{Z}_2^n$ is a $1 \times n$ binary vector with a given number $t \ll n$ of non-zero entries named error vector, representing the unknown sequence of values to be found, while H' is assumed to have $d_v \ll n$ non-zero entries per column. Therefore, the purpose of an iterative decoding procedure is to compute the values of the elements of e given H' and s.

A common approach to perform iterative decoding is the *Bit Flipping* (BF) strategy firstly described in [13]. Such an approach considers the i-th row of H', with $i \in \{0, \ldots, r-1\}$, as a representation of the coefficients of the parity check equation involving the unknown e_j, with $j \in \{0, \ldots, n-1\}$, having as constant term the i-th element of the syndrome s. Each coefficient is associated to a binary variable $e_j \in \mathbb{Z}_2$, i.e., a binary element of the error vector e whose value should be determined. Initially, the guessed value of the error vector, denoted in the following as $\hat{e}$, is assumed to be the null vector, i.e., $\hat{e} = 0_{1\times n}$ (i.e., the bits of the received message are initially assumed to be all uncorrupted).

The iterative BF decoding procedure repeats (at least one time) the execution of two phases (named in the following as *Count of the unsatisfied parity checks*, and *Bit-flipping*, respectively) until either all the values of the syndrome become null (pointing out the fact that every value of e has been found) or an imposed a-priori maximum number of iterations, $l_{\max} \geq 1$, is reached.

1. *Count of the unsatisfied parity checks.* The first phase of the decoding procedure analyzes the parity check equations where a given error variable $\hat{e}_j$ is involved, with $j \in \{0, \ldots, n-1\}$, i.e., the number of rows of H' where the j-th element is non-zero, and counts how many of them are unsatisfied. In other words, it counts how many equations involving the unknown e_j have a constant term in the syndrome which is non-zero. Such a count of the number of unsatisfied parity check equations, $\mathtt{upc}_j$, can be computed for each error variable $\hat{e}_j$, lifting the elements of s and H' from $\mathbb{Z}_2$ to $\mathbb{Z}$ and performing a product between an integer vector ($\varsigma \leftarrow \mathtt{Lift}(s)$) by an integer matrix ($\mathcal{H}' \leftarrow \mathtt{Lift}(H')$), obtaining a $1 \times n$ integer vector $\mathtt{upc}^{(\mathrm{BF})}$, i.e., $\mathtt{upc}^{(\mathrm{BF})} \leftarrow \varsigma\, \mathcal{H}'$.

2. *Bit-flipping.* The second phase changes (i.e., flips, hence the name bit-flipping) each value of an error variable $\hat{e}_j$ for which $\mathtt{upc}_j^{(\mathrm{BF})}$ exceeds a given threshold $b \geq 1$. Subsequently, it updates the value of the syndrome, computing it as $H'\hat{e}^T$, employing the new value of the $\hat{e}_j$ variables in the process.

The LEDA cryptosystems leverage Q-decoders that achieve smaller complexity than classical BF decoders. In fact, due to the sparsity of both H and Q, their product HQ has a number of non-zero row elements $\leq d_c\, m$, with the equality holding with very high probability. Such a fact can be exploited to perform the first phase of the bit-flipping decoding procedure in a more efficient way. To do so, the Q-decoder proceeds to lift H and Q in the integer domain obtaining $\mathcal{H} \leftarrow \mathtt{Lift}(H)$ and $\mathcal{Q} \leftarrow \mathtt{Lift}(Q)$, respectively. Subsequently, it performs a decoding strategy similar to the one described above, as follows.

1. *Count of the unsatisfied parity checks.* The first phase is performed in two steps. First of all, a temporary $1 \times n$ vector of integers $\mathtt{upc}^{(\mathrm{temp})}$ is computed in the same fashion as in the BF decoder, employing the lifted syndrome, $\varsigma \leftarrow \mathtt{Lift}(s)$, and $\mathcal{H}$ instead of $\mathcal{H}'$, i.e., $\mathtt{upc}^{(\mathrm{temp})} \leftarrow \varsigma\, \mathcal{H}$. The value of the actual $1 \times n$ integer vector $\mathtt{upc}^{(\mathrm{Q-dec})}$ storing the unsatisfied parity-check counts is then computed as: $\mathtt{upc}^{(\mathrm{Q-dec})} \leftarrow \mathtt{upc}^{(\mathrm{temp})}\ \mathcal{Q}$.

2. *Bit-flipping.* The second phase of the Q-decoder follows the same steps of the BF one, flipping the values of the guessed error vector $\hat{e}_j$, $j \in \{0, \ldots, n-1\}$, for which the j-th unsatisfied parity-check count $\mathtt{upc}_j^{(\mathrm{Q-dec})}$ exceeds the chosen threshold b. Subsequently, the value of the syndrome s is updated as $s + HQ\hat{e}^T$.

In both the BF- and Q-decoder, the update to the syndrome value caused by the flipping of the values of $\hat{e}$ in the second phase of the procedure, can be computed incrementally, adding only the contributions due to the value change of $\hat{e}$ (see the LEDA cryptosystems specification [3]).

If a null s is obtained before the maximum allowed number of iterations $l_{\max}$ is exceeded, then the Q-decoder terminates with success its decoding procedure, otherwise it ends with a decoding failure.

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

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

hence $\mathtt{upc}_j^{(\mathrm{BF})} = \sum_{z=0}^{r-1} h'_{zj}\, s_z$. The computation of $\mathtt{upc}^{(\mathrm{Q-dec})}$ proceeds as follows

$$\mathtt{upc}^{(\mathrm{Q-dec})} = (\varsigma\, \mathcal{H})\, \mathcal{Q} = \sum_{i=0}^{n-1} \left(\sum_{z=0}^{r-1} s_z\, h_{zi} \right) q_{ij} = \sum_{z=0}^{r-1} \left(\sum_{i=0}^{n-1} h_{zi}\, q_{ij} \right) s_z.$$

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

Lemma 2 (Computational advantage of the Q-decoder). *Let us consider a bit-flipping decoding procedure and a Q-decoder procedure both acting on the same parity matrix $H' = HQ$. The number of non-zero entries of a column of H is $d_v \ll n$, the number of non-zero entries of a column of Q is $m \ll n$, and the number of non-zero entries of a column of H' is $d_v m$ (assuming no cancellations occur in the multiplication HQ). The computational complexity of an iteration of the bit-flipping decoder equals $\mathcal{O}(d_v mn + n)$, while the computational complexity of an iteration of the Q-decoder procedure is $\mathcal{O}((d_v + m)n + n)$.*

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

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

The decoding failure rate of the Q-decoder is crucially dependent on the choice made for the bit-flipping threshold b. Indeed, the designer aims at picking a value of b satisfying the following criteria.

(i) The number of values of the unsatisfied parity-check count $\mathtt{upc}_j$, with $j \in \{0, \ldots, n-1\}$ such that $\hat{e}_j \neq e_j$ and $\mathtt{upc}_j \geq b$ should be maximum. Indeed, when this situation occurs, all such values $\mathtt{upc}_j$ are rightfully flipped.

(ii) The number of values of the unsatisfied parity-check count $\mathtt{upc}_j$, with $j \in \{0, \ldots, n-1\}$ such that $\hat{e}_j = e_j$ and $\mathtt{upc}_j < b$ should be maximum. Indeed, when this situation occurs, all such values $\mathtt{upc}_j$ are rightfully not flipped.

Observing that during the decoding procedure the j-th bit value of the guessed error vector $\hat{e}$ is flipped when $\mathtt{upc}_j$ is higher than or equal to b, an ideal case where it is possible to attain a null DFR is the one in which, whenever the maximum possible value of any unsatisfied parity-check count $\mathtt{upc}_j$ related to a variable $\hat{e}_j = e_j$ (i.e., no flip is needed) is lower than the minimum possible value of $\mathtt{upc}_k$ related to any variable $\hat{e}_k \neq e_k$ (i.e., flip is needed). Indeed, in this case, setting the threshold to any value b such that $\mathtt{max\_upc}_{no\,flip} < b \leq \mathtt{min\_upc}_{flip}$ allows the Q-decoder to compute the value of the actual error vector e in a single iteration.

To provide code parameter design criteria to attain a zero DFR in a single iteration with the Q-decoder, we now analyze the contribution to the values of $\mathtt{upc}$ provided by the bits of the actual error vector. Let $u_z \in \mathbb{Z}_2^n$, with $z \in \{0, \ldots, n-1\}$, denote $1 \times n$ binary vector such that only the z-th component of u_z is non-zero (i.e., it has unitary Hamming weight, $wt(u_z) = 1$). We now consider the actual error vector e as the sum of $t \geq 1$ vectors $\in \mathbf{U}(e) = \{u_z \in \mathbb{Z}_2^n,\ wt(u_z) = 1, z \in \mathbf{I}(e)\}$, where $\mathbf{I}(e) \subset \{0, \ldots, n-1\}$ defines the support of e and $|\mathbf{I}(e)| = t$ (thus it also holds $|\mathbf{U}(e)| = t$), and quantify the contributions of each bit in e to the value of $\mathtt{upc}_z$ computed by the Q-decoder in its first iteration, proceeding backwards from each $u_z \in \mathbf{U}(e)$ composing e.

We describe the mentioned quantification with the aid of a running example referred to the syndrome Q-decoding procedure of a toy code $\mathcal{C}(5, 3, 2)$, assuming that a single bit in the actual error vector is asserted, i.e., $e = u_2$. Figure 2 reports a graphical description of the mentioned running example. Our aim is to define a $d_v m \times n$ matrix $P_{(z)}$ containing a set of parity-check equations, i.e., rows of H which contribute to $\mathtt{upc}_z$, $z \in \{0, \ldots, n-1\}$[6].

Consider the syndrome value $s = \left(H(Qe^T)\right)^T$ obtained as the multiplication between the matrix Q and the actual error vector value $e = u_z$, with $z \in \{0, \ldots, n-1\}$, (i.e., $Q\,e^T = Q\,u_z^T$), followed by the multiplication between the matrix H and the *expanded error* vector $\mathtt{e}^{(z)} = Q\,e^T$, with $wt(\mathtt{e}^{(z)}) = m$, which has been computed in the previous step (i.e., $s = H\left(\left(\mathtt{e}^{(z)}\right)^T\right)^T$. Note that, in this case $\mathtt{e}^{(z)}$ is the result of a computation depending only on the value of Q and z, not on the actual error vector being decoded.

Consider $\mathtt{e}^{(z)}$ as the sum of m binary vectors u_j, $j \in \{0, \ldots, n-1\}$ with a single non-zero entry in the j-th position (see $\mathtt{e}^{(2)} = u_2 + u_4$, with u_2 highlighted

[6] Note that the notation $P_{(z)}$ denotes a matrix whose values are related with the bit in position z of the actual (unknown) error vector (it may include repeated rows). The round brackets employed in the subscript are meant to disambiguate this object from the notations related to the z-th row of a generic matrix P, i.e., P_z.

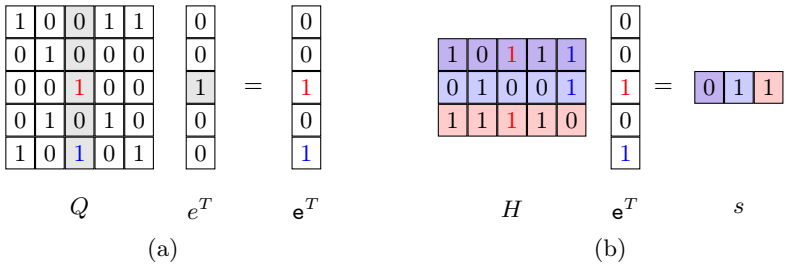

Fig. 2. Steps of the syndrome computation process of a toy code $\mathcal{C}(5,3,2)$, having H with constant column weight $d_v = 2$ and Q with constant column weight $m = 2$. The single bit error vector employed is $e = u_2 = (0,0,1,0,0)$. In (a) the error vector is expanded as $\mathsf{e} = u_2 + u_4 = (0,0,1,0,1)$ after the multiplication by Q, $\mathsf{e} = (Qe^T)^T$. In (b) the effect on the syndrome of multiplying e by H is shown, i.e., $s = \left(H(Qe^T)\right)^T$.

in red and u_4 in blue in Fig. 2). Each u_j in $\mathsf{e}^{(z)}$ is involved in all the parity-check equations of H, i.e., the $1 \times n$ rows, H_i, with $i \in \{0, \ldots, r-1\}$, having their j-th element set to 1. It is thus possible to build a $d_v m \times n$ matrix $P_{(z)}$ juxtaposing all the parity equations in H such that their j-th element is set to 1, for all the u_j composing $\mathsf{e}^{(z)}$. The $P_{(z)}$ matrix allows to compute the contribution to the value $\mathtt{upc}_z$, $z \in \{0, \ldots, n-1\}$ provided by any expanded error vector $\mathsf{e}^{(j)}$, $j \in \{0, \ldots, n-1\}$, as it is constituted by all and only the parity check equations which will have their non null results counted to obtain $\mathtt{upc}_z$. Therefore, for any binary variable e_z, $z \in \{0, \ldots, n-1\}$, in the actual error vector e, it is possible to express the value of the corresponding unsatisfied parity-check count evaluated by the decoding procedure as $\mathtt{upc}_z \leftarrow wt\left(P_{(z)}\left(\mathsf{e}^{(z)}\right)^T\right)$.

The construction of $P_{(z)}$ for the toy example is reported in Fig. 3, where $z = 2$. $P_{(z)}$ is obtained by juxtaposing the rows H_0 and H_2, as they both involve u_2, and juxtaposing to them the rows H_0 and H_1 as they both involve u_4.

Figure 3 provides a visual comparison of the two equivalent processes to compute the value of $\mathtt{upc}_2$ considering the values of $\varsigma, \mathcal{H}$ and $\mathcal{Q}$ obtained as the integer lifts of s, H and Q from Fig. 2. Indeed, computing the value of $\mathtt{upc}_2$ as the third component of the vector $\varsigma\mathcal{H}\mathcal{Q}$ yields the same results as computing the binary vector $P_{(2)}(\mathsf{e}^{(2)})^T$ and computing its weight.

Relying on the $P_{(z)}$ matrices to express the contribution of a given expanded error vector $\mathsf{e}^{(z)}$, we are able to rewrite the computation of $\mathtt{upc}_z$ for a generic error vector with t asserted bits, i.e., $e = \sum_{u_z \in \mathbf{U}(e)} u_z$, where $\mathbf{I}(e) \subset \{0, \ldots, n-1\}$ with $|\mathbf{I}(e)| = t$, and $\mathbf{U}(e) = \{u_i,\ wt(u_i) = 1,\ i \in \mathbf{I}(e)\}$, as follows

$$\mathtt{upc}_z \leftarrow wt\left(\sum_{i \in \mathbf{I}(e) \subset \{0,\ldots,n-1\}} P_{(z)}\left(\mathsf{e}^{(i)}\right)^T\right) = wt\left(\sum_{u_z \in \mathbf{U}(e)} P_{(z)}(Q\,u_z^T)\right).$$

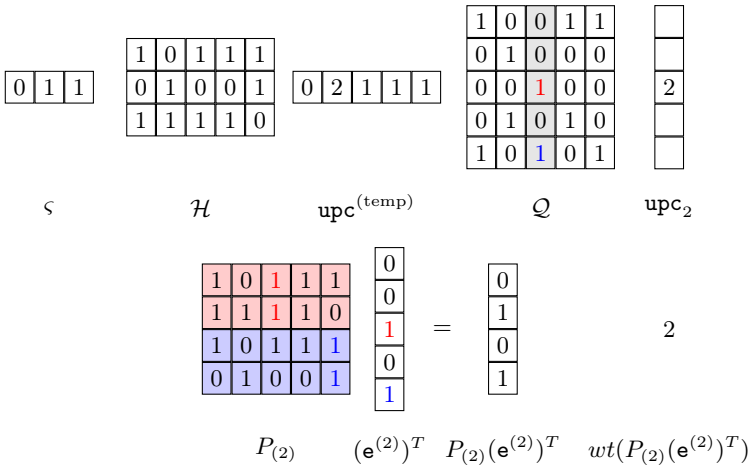

Fig. 3. Representation of the $P_{(z)}$ matrix for the running example ($z = 2$), and computation of the $\mathtt{upc}_z$ value both via $\mathcal{Q}$ and $\mathcal{H}$, and employing $P_{(z)}$

A.1 Q-decoders with zero DFR

Having provided a way to derive the contribution of any bit of an actual error vector to a specific unsatisfied parity-check count $\mathtt{upc}_z$, following the work in [37] for the BF-decoder, we now proceed to analyze the case of a Q-decoder which is always able to correct all the errors in a single iteration of the decoding procedure. To do so, the bit-flipping action should flip the value of all the elements of the guessed error $\hat{e}$ which do not match e. Recalling that $\hat{e}$ is initialized to the null vector, the first iteration of the Q-decoder should thus flip all the elements $\hat{e}_z$ such that $e_z = 1$.

The Q-decoder will perform all and only the appropriate flips if the $\mathtt{upc}_z$ with $z \in \{0, \ldots, n-1\}$ such that $e_z = 1$, match or exceed the threshold b, and all the $\mathtt{upc}_z$ such that $e_z = 0$ are below the same flipping threshold.
We have that, if the highest value of $\mathtt{upc}_z$ when $e_z = 0$ (i.e., $\mathtt{max\_upc}_{no\,flip}$) is smaller than the lowest value of $\mathtt{upc}_z$ when $e_z = 1$ (i.e., $\mathtt{min\_upc}_{flip}$), the Q-decoder will be able to correct all the errors in a single iteration if the bit-flipping threshold b is set to $b = \mathtt{min\_upc}_{flip}$, as this will cause the flipping of all and only the incorrectly estimated bits in the guessed error vector $\hat{e}$.

In the following, we derive an upper bound on the maximum admissible number of errors t which guarantees that $\mathtt{max\_upc}_{no\,flip} < \mathtt{min\_upc}_{flip}$ for a given code.

Theorem 1. *Let H be an $r \times n$ parity-check matrix, with constant column weight equal to d_v, and let Q be an $n \times n$ matrix with constant row weight equal to m. Let e be a $1 \times n$ binary error vector with weight t, composed as $e = \sum_{i \in \mathbf{I}(e)} u_i$, with $\mathbf{I}(e) \subset \{0, \ldots, n-1\}$ defining the support of e, $|\mathbf{I}(e)| = t$, where $u_i \in \mathbb{Z}_2^n$, and*

$wt(u_i) = 1$; and let $\mathtt{e}$ be the $1 \times n$ binary vector computed as $\mathtt{e} = \left(Qe^T\right)^T$. The first iteration of the Q-decoder, taking as input the $1 \times r$ syndrome $s = (H\mathtt{e}^T)^T$, retrieves the values of all the bits of actual error vector e if $t < \frac{\alpha+\beta}{\gamma+\beta}$, where

$$\begin{cases} \alpha = \min_{z \in \mathbf{I}(e)} \left\{ wt\left(P_{(z)}\left(\mathtt{e}^{(z)}\right)^T\right)\right\}, \\ \beta = \max_{z,i \in \mathbf{I}(e), z \neq i} \left\{ wt\left(P_{(z)}\left(\mathtt{e}^{(z)}\right)^T \wedge P_{(z)}\left(\mathtt{e}^{(i)}\right)^T\right)\right\}, \\ \gamma = \max_{z,i \in \mathbf{I}(e), z \neq i} \left\{ wt\left(P_{(z)}\left(\mathtt{e}^{(i)}\right)^T\right)\right\}, \end{cases} \tag{1}$$

where $\wedge$ indicates the component-wise binary product (i.e., the Boolean `and`).

Proof. We first determine the lower bound $\mathtt{min\_upc}_{flip}$ as the lowest value of an unsatisfied parity-check count $\mathtt{upc}_z$, $z \in \{0, \ldots, n-1\}$, when the value of the corresponding bit in the actual error vector is set, i.e., $e_z = 1$.
We start by decomposing the value of $\mathtt{upc}_z$ into the contributions provided by the t vectors $u_i \in \mathbb{Z}_2^n$, with $i \in \{0, \ldots, n-1\}$ and $wt(u_i) = 1$, i.e., $e = \sum_{i \in \mathbf{I}(e)} u_i$, $\mathbf{I}(e) \subset \{0, \ldots, n-1\}, |\mathbf{I}(e)| = t$ (thus, $z \in \mathbf{I}(e)$). In other words, we want to quantify the minimum value for the count of the unsatisfied parity checks related to a bit of the guessed error vector that needs to be flipped when it is right to do so. We then have

$$\mathtt{upc}_z = wt\left(P_{(z)}\ \mathtt{e}^T\right) = wt\left(P_{(z)}\left(\mathtt{e}^{(z)}\right)^T \oplus \bigoplus_{\substack{\mathtt{e}^{(i)} = (Qu_i^T)^T \\ u_i \in \mathbf{U}(e) \setminus \{u_z\}}} P_{(z)}\left(\mathtt{e}^{(i)}\right)^T\right).$$

Considering that, for a generic pair of binary vectors a, b of length n we have that $wt(a \oplus b) = wt(a) + wt(b) - 2wt(a \wedge b)$, we expand the former onto

$$\mathtt{upc}_z = wt\left(P_{(z)}\left(\mathtt{e}^{(z)}\right)^T\right) + wt\left(\bigoplus_{\substack{\mathtt{e}^{(i)} = (Qu_i^T)^T \\ u_i \in \mathbf{U}(e) \setminus \{u_z\}}} P_{(z)}\left(\mathtt{e}^{(i)}\right)^T\right) +$$
$$- 2\ wt\left(P_{(z)}\left(\mathtt{e}^{(z)}\right)^T \bigwedge \bigoplus_{\substack{\mathtt{e}^{(i)} = (Qu_i^T)^T \\ u_i \in \mathbf{U}(e) \setminus \{u_z\}}} P_{(z)}\left(\mathtt{e}^{(i)}\right)^T\right).$$

Recalling that, for two binary vectors a, b it holds that $wt(b) \geq wt(a \wedge b)$, it is worth noting that it also holds that $wt(a) + wt(b) - 2\,wt(a \wedge b) \geq wt(a) - wt(a \wedge b)$. Therefore, considering the second and third addend of the above equality on

$\mathtt{upc}_z$, we obtain

$$\mathtt{upc}_z \geq wt\left(P_{(z)}\left(\mathbf{e}^{(z)}\right)^T\right) - wt\left(P_{(z)}\left(\mathbf{e}^{(z)}\right)^T \bigwedge \bigoplus_{\substack{\mathbf{e}^{(i)}=(Qu_i^T)^T \\ u_i \in \mathbf{U}(e)\setminus\{u_z\}}} P_{(z)}\left(\mathbf{e}^{(i)}\right)^T\right).$$

Since we are interested in quantifying the lower bound $\mathtt{min\_upc}_{flip}$ for $\mathtt{upc}_z$, a straightforward rewriting of the previous inequality is as follows

$$\begin{aligned}\mathtt{upc}_z \geq \quad & \min_{z\in\mathbf{I}(e)}\left\{wt\left(P_{(z)}\left(\mathbf{e}^{(z)}\right)^T\right)\right\} + \\ & - \max_{\substack{z,i\in\mathbf{I}(e)\\ z\neq i}}\left\{wt\left(P_{(z)}\left(\mathbf{e}^{(z)}\right)^T \bigwedge \bigoplus_{\substack{\mathbf{e}^{(i)}=(Qu_i^T)^T \\ u_i \in \mathbf{U}(e)\setminus\{u_z\}}} P_{(z)}\left(\mathbf{e}^{(i)}\right)^T\right)\right\}.\end{aligned}$$

Considering the second addend, a coarser upper bound to the argument of the max{...} operator can be derived observing that, given three binary vectors a, b, c, $wt\,(c \wedge (a \oplus b)) \leq wt\,(c \wedge (a \vee b)) = wt\,((c \wedge a) \vee (c \wedge b))$, where $\vee$ denotes the binary or. Thus, the second addend in the previous inequality can be replaced by the following quantity

$$\max_{\substack{z,i\in\mathbf{I}(e)\\ z\neq i}}\left\{wt\left(\bigvee_{\substack{\left(\mathbf{e}^{(i)}\right)=(Qu_i^T)^T \\ u_i \in \mathbf{U}(e)\setminus\{u_z\}}}\left(P_{(z)}\left(\mathbf{e}^{(z)}\right)^T \bigwedge P_{(z)}\left(\mathbf{e}^{(i)}\right)^T\right)\right)\right\}$$

which can be further upper bounded (noting that $|\mathbf{U}(e) \setminus \{u_z\}| = t-1$) as

$$(t-1)\max_{\substack{z,i\in\mathbf{I}(e)\\ z\neq i}}\left\{wt\left(P_{(z)}\left(\mathbf{e}^{(z)}\right)^T \bigwedge P_{(z)}\left(\mathbf{e}^{(i)}\right)^T\right)\right\}$$

Looking at the original equality set to compute the value of $\mathtt{upc}_z$, it holds that

$$\begin{aligned}\mathtt{upc}_z \geq \quad & \min_{z\in\mathbf{I}(e)}\left\{wt\left(P_{(z)}\left(\mathbf{e}^{(z)}\right)^T\right)\right\} + \\ & - (t-1)\max_{\substack{z,i\in\mathbf{I}(e)\\ z\neq i}}\left\{wt\left(P_{(z)}\left(\mathbf{e}^{(z)}\right)^T \bigwedge P_{(z)}\left(\mathbf{e}^{(i)}\right)^T\right)\right\}\end{aligned}$$

Therefore, it is easy to acknowledge that

$$\mathtt{min\_upc}_{flip} \geq \alpha - (t-1)\beta.$$

In the following we determine $\mathtt{max\_upc}_{no\,flip}$ as the highest value of an unsatisfied parity-check count $\mathtt{upc}_z$, $z \in \{0, \ldots, n-1\}$, when the value of the corresponding bit in the actual error vector is unset, i.e. $e_z = 0$.

In this case we aim at computing an upper bound for $\mathtt{upc}_z$, considering that that $u_z \notin \mathbf{U}(e)$ (and thus $z \notin \mathbf{I}(e)$); we have

$$\mathtt{upc}_z = wt\left(\bigoplus_{\substack{\mathtt{e}^{(i)}=(Qu_i^T)^T \\ u_i \in \mathbf{U}(e)}} P_{(z)}\left(\mathtt{e}^{(i)}\right)^T\right) \leq t \max_{z,i \in \mathbf{I}(e)} \left\{wt\left(P_{(z)}\left(\mathtt{e}^{(i)}\right)^T\right)\right\} = t\gamma$$

We can therefore employ $t\gamma$ as an upper bound for the value of $\mathtt{max\_upc}_{no\,flip}$.

Recalling that the Q-decoder procedure will retrieve all the values of the error vector e in a single iteration if $\mathtt{max\_upc}_{no\,flip} < \mathtt{min\_upc}_{flip}$ and the bit-flipping threshold b is such that $b = \mathtt{min\_upc}_{flip}$, it is easy to acknowledge that the maximum number of errors tolerable by the code is constrained by the following inequality

$$t\gamma < \alpha - (t-1)\beta \Rightarrow t < \frac{\alpha+\beta}{\gamma+\beta}. \tag{2}$$

□

A.2 Probabilistic Analysis of the First Iteration of the Q-Decoder

In the following, we model the number of differences between the guessed error vector $\hat{e}$, provided as output of the first iteration of the Q-decoder, and the actual error vector e, as a random variable $\mathtt{T}$ over the discrete domain of integers $\{0, \ldots, t\}$, $t \geq 0$ having a probability mass function $Pr[\mathtt{T} = \tau]$, $\tau = wt(\hat{e}^* \oplus e)$ depending on the decoding strategy and the LDPC code parameters.

To quantify the said probability, we consider the decoding procedure employed by the LEDAcrypt systems assuming that the hypothesis of Lemma 1 holds. Given the equivalence of the BF decoder and Q-decoder provided by this Lemma, for the sake of simplicity, we will reason on the application of one iteration of the BF decoder taking as input the $r \times n$ parity-check matrix $H' = HQ$ (assumed to be computed as a cancellation-free product between H and Q), the $1 \times n$ syndrome $s = (H'\,\hat{e}^T)^T$, and a null guessed error vector $\hat{e} = 0_{1\times n}$. The code is assumed to be an LDPC code as in the LEDA cryptosystems, with $r = (n_0 - 1)p$, $n = n_0 p$, p a prime number, $n_0 \in \{2, 3, 4\}$, while m denotes the number of non-zero entries in each row/column of the $n \times n$ matrix Q, and $d_v\, n_0$ denotes the number of non-zero entries in each row/column of the $r \times n$ matrix H. As a consequence, each row/column of the parity-check matrix H' exhibits $d'_c = d_v\, n_0\, m$ non-zero entries. This implies that each parity-check equation (i.e., row) of H' involves d'_c variables of the guessed error vector $\hat{e}$.

The quantification of the probability to observe a certain number differences between the guessed error vector, provided as output of the first iteration of the decoder, and the actual error vector can be evaluated considering the number of correctly and wrongly flipped bits after the first iteration of the decoding

algorithm. In turn, each of these numbers, can be quantified reasoning on the following joint probabilities: $\mathtt{p_{correct-unsatisfied}}$ and $\mathtt{p_{incorrect-unsatisfied}}$.

The joint probability $\mathtt{p_{correct-unsatisfied}} = Pr\left[\hat{e}_j = e_j = 0;\, h_{ij} = 1,\, s_i = 1\right]$ can be stated as the likelihood of occurrence of the following events:

- $\{\hat{e}_j = e_j = 0\}$ refers to the event describing the j-th bit of the guessed error vector that does not need to be flipped;
- $\{h_{ij} = 1,\, s_i = 1\}$ refers to the event describing the i-th parity-check equation (i.e., H'_i) that is unsatisfied (i.e., $s_i = 1$) when the j-th variable in $\hat{e}$ (i.e., $\hat{e}_j$) is included in it.

It is easy to acknowledge that the above events occur if an odd number of the t asserted bits in the unknown error vector are involved in $d'_c - 1$ parity-check equations, thus

$$\mathtt{p_{correct-unsatisfied}} = \sum_{j\,=\,1,\,j\ \mathrm{odd}}^{\min\left[d'_c-1,t\right]} \frac{\binom{d'_c-1}{j}\binom{n-d'_c}{t-j}}{\binom{n-1}{t}}.$$

An analogous line of reasoning allows to quantify also the joint probability $\mathtt{p_{incorrect-unsatisfied}} = Pr\left[\hat{e}_i \neq e_i;\, h_{ij} = 1,\, s_i = 1\right]$, which can be stated as the likelihood of occurrence of the following events:

- $\{\hat{e}_j \neq e_j\}$ refers to the event describing the j-th bit of the guessed error vector that need to be flipped;
- $\{h_{ij} = 1,\, s_i = 1\}$ refers to the event describing the i-th parity-check equation (i.e., H'_i) that is unsatisfied (i.e., $s_i = 1$) when the j-th variable in $\hat{e}$ (i.e., $\hat{e}_j$) is included in it. So

$$\mathtt{p_{incorrect-unsatisfied}} = \sum_{j\,=\,0,\,j\ \mathrm{even}}^{\min\left[d'_c-1,t-1\right]} \frac{\binom{d'_c-1}{j}\binom{n-d'_c}{t-j-1}}{\binom{n-1}{t-1}}.$$

The probability $\mathtt{p_{correct}}$ that the `upc` based estimation deems rightfully a given $\hat{e}_j \neq e_j$ in need of flipping can be quantified as the probability that $\mathtt{upc} \geq b$, i.e.,

$$\mathtt{p_{correct}} = \sum_{j=b}^{d'_v} \binom{d'_v}{j} \mathtt{p_{incorrect-unsatisfied}}^{j} \left(1 - \mathtt{p_{incorrect-unsatisfied}}\right)^{d'_v - j}.$$

Analogously, we define the probability $\mathtt{p_{induce}}$ as the probability that the `upc` based estimation deems a given $\hat{e}_j = e_j$ as (wrongly) in need of flipping as

$$\mathtt{p_{induce}} = \sum_{j=b}^{d'_v} \binom{d'_v}{j} \mathtt{p_{correct-unsatisfied}}^{j} \left(1 - \mathtt{p_{correct-unsatisfied}}\right)^{d'_v - j}.$$

Note that $\mathtt{p_{correct}}$ is indeed the probability that the Q-decoder performs a correct flip at the first iteration, while $\mathtt{p_{induce}}$ is the one of performing a wrong flip.

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

$$Pr\left[f_{\mathtt{correct}} = c\right] = \binom{t}{c} {\mathtt{p}_{\mathtt{correct}}}^{c} \left(1 - \mathtt{p}_{\mathtt{correct}}\right)^{t-c},$$
$$Pr\left[f_{\mathtt{wrong}} = w\right] = \binom{n-t}{w} {\mathtt{p}_{\mathtt{induce}}}^{w} \left(1 - \mathtt{p}_{\mathtt{induce}}\right)^{n-t-w}.$$

Assuming that the decision on whether a given value $\hat{e}_j$ in $\hat{e}$ should be flipped or not are taken independently, i.e.,

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

we obtain the probability that the guessed error vector $\hat{e}$, at the end of the computation of the first iteration of the Q-decoder, differs from the actual error vector in $\tau \in \{0, \ldots, t\}$ positions as follows

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

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