Design of LEDAkem and LEDApkc instances with tight parameters and 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 two interrelated code-based cryptosystems: the key encapsulation mechanism LEDAkem and the public-key cryptosystem LEDApkc, which are among the round 1 submissions to the NIST post-quantum cryptography project.
We provide a detailed quantification of the quantum and classic computational effort levels needed to foil the cryptographic guarantees of these systems. To this end, we take into account the best 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.
We introduce an algorithmic optimization procedure to design new sets of parameters for LEDAkem and LEDApkc and make the corresponding software implementation publicly available. We report novel parameter sets for LEDAkem and LEDApkc that 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 LEDA 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 someone decrypting ciphertexts with the same private key, and consequentially it is able to guarantee a lifecycle of the corresponding key pairs which can be sufficient for the wide majority of practical purposes.

1 Introduction

This document reports theoretical and implementation advancements concerning two cryptosystems that have been admitted as round 1 candidates to the NIST call for post-quantum cryptographic systems [24], named LEDAkem (low density parity-check code-based key encapsulation mechanism) and LEDApkc (low-density parity-check code-based public-key cryptosystem).

The mathematical problem on which these systems rely is the one of decoding a random-looking linear block code, which is one of the problems whose solution cannot exploit any known polynomial time algorithm on a quantum computer. Such a problem in fact belongs to the class of NP-complete problems [5,16], which is widely believed to contain problems without polynomial time solution on a quantum computer. This line of research was initiated by McEliece in 1978 [20], using Goppa codes as secret codes, and Niederreiter in 1986 [25], with a first attempt of introducing generalized Reed-Solomon (GRS) codes in this 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 [10], low-density parity-check (LDPC) codes [23], quasi-dyadic (QD) codes [21], quasi-cyclic low-density parity-check (QC-LDPC) codes [3] and quasi-cyclic moderate-density parity-check (QC-MDPC) codes [22].

The distinguishing points of the LEDAkem and LEDApkc cryptosystems with respect to other code-base 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 in [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, which are algorithms for decoding a general linear block code. In addition, a recent class of 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 life length of keypairs used in code-based cryptosystems characterized by a non-zero DFR [7,8,14]. In the following sections we analyse all the aforementioned attacks and detail how to tune the parameter design of LEDA cryptosystems to foil them.

Contribution. The contribution of this manuscript can be summarized as follows. *(i)* A quantification of the quantum and classic computational effort levels we considered as the computational requirements to break AES. We relied on classical circuit design estimates for the classical computing complexity, and on the work by Grassl et al. at PQCrypto 2016 [12] for the quantum computing complexity. *(ii)* The description of a new algorithmic approach to the design of LEDAkem and LEDApkc instances with optimal and tight parameters, based on the NIST requirements. The proposed approach employs estimations of the computational efforts required to perform Information Set Decoding attacks in the finite regime (as opposed to employing asymptotic bounds) as well as executing an exhaustive search in the parameter space of the algorithms. *(iii)* New optimal parameters of the LEDA cryptosystems matching the NIST security requests, which allows us to show running time and key size figures exhibiting a $\times 3.5$–$\times 6.8$ speedup on the original reference implementation and a $\approx \times 2$ key size reduction w.r.t. the original submission parameters. *(iv)* A novel technique allowing us to design a set of QC-LDPC code parameters for use in LEDAkem/LEDApkc deriving an upper bound to the code DFR in closed-form. This allows to include a bound on the DFR as a parameter design criterion. Finally, we report sample sets of parameters targeting an upper bound for the DFR of 2^{-64} for long term keys in LEDApkc.

Preliminaries and notation. LEDAkem and LEDApkc leverage the same theoretical framework, the main difference being that LEDAkem implements a Niederreiter-like cryptosystem, while LEDApkc follows the McEliece approach. In addition, LEDApkc implements a conversion which allows using public generator matrices in systematic form and to achieve security against a chosen ciphertext attack (CCA) under the assumption of negligible DFR. Since these modifications make the description of LEDApkc more involved than that of LEDAkem, we will follow the conventions and notation of the latter for the sake of simplicity. Nevertheless, our analysis can be equally applied to either LEDAkem or LEDApkc.

LEDAkem [2] 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 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 matrices. Their product $H' = HQ$ still gives a sparse matrix 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. Concerning the error correction capability of these codes, let us remind that QC-LDPC codes are decoded through iterative algorithms that are not bounded-distance decoders. Therefore, their decoding radius is not deterministic and the decoding failure rate is bounded away from zero. We denote as $t \ll n$ the number of errors that can be corrected by the code defined by H with a sufficiently large probability, and the code itself is denoted as $C(n, k, t)$. Given t, encryption starts with mapping of the secret message (or part of it) into a random binary vector e with length of n bits and Hamming weight t. Indeed, the secret message in LEDAkem is a randomly generated key, thus e is randomly generated. Then, a syndrome of e is computed through the public parity-check matrix and this gives the ciphertext. Decryption starts by performing syndrome decoding through the private code, which allows recovering the expanded error vector $e' = eQ^T$ (apart from some DFR). Then e is recovered from e' through multiplication by the inverse of Q.

Paper organization. The document is organized as follows. In Section 2 we define the security level benchmarks we consider, in compliance with the NIST requirements. In Section 3 we describe the attacks we take into account into the design approach we propose. In Section 4 we describe an algorithmic procedure, publicly available in software, for the design of optimization of sets of parameters of these systems. In Section 5 we provide a new parametrization of these systems which includes tighter, optimal choices of the system parameters. In Section 6 we introduce a theoretical characterization of the DFR of LEDAkem and LEDApkc, and report parameter sets which guarantee a DFR lower than 2^{-64}.

2 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 Advanced Encryption Standard (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. Throughout the design of the parameters for the LEDA cryptosystems we ignore Categories 2 and 4: if a cipher matching those security levels is required, we advise to employ the parameters for Categories 3 and 5, respectively.

The computational worst-case complexity of breaking AES on a classical computer can be estimated as $2^{\lambda}\mathtt{C}_{\mathtt{AES}}$, where $\mathtt{C}_{\mathtt{AES}}$ is the amount of binary operations required to compute AES on a classical computer on a small set of plaintexts, and match them with a small set of corresponding ciphertexts to validate the correct key retrieval. Indeed, more than a single plaintext-ciphertext pair is required to retrieve AES keys [12]. 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 [30], 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 [30], 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 [12], where a detailed implementation of an AES breaker is provided. The computation considers an implementation of Grover's algorithm [13] 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 [12] 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

Table 1. Classical and quantum computational costs to perform an exhaustive key search on AES

NIST Category	AES Key Size (bits)	Classical Cost (binary operations)	Quantum Cost [12] (quantum gates)
1	128	$2^{128} \cdot 2^{14} \cdot 3 = 2^{143.5}$	$1.16 \cdot 2^{81}$
3	192	$2^{192} \cdot 2^{14} \cdot 4 = 2^{208}$	$1.33 \cdot 2^{113}$
5	256	$2^{256} \cdot 2^{14} \cdot 5 = 2^{272.3}$	$1.57 \cdot 2^{145}$

the figures reported in [12] 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.

3 Evaluated attacks

With reference to the LEDA cryptosystems specification [1], 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, like 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 available at `https://github.com/ledacrypt`.

3.1 Attacks based on exhaustive key search

Enumerating all the possible values for the secret key is, in principle, applicable to any cryptosystem. The original LEDA cryptosystems specification documents [1] do not mention exhaustive key search, as they are strictly dominated by other, less computationally demanding, attack strategies such as the use of Information Set Decoding (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 LEDA cryptosystems 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.

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, reported as $\{m_0, \dots, m_{n_0-1}\}$ in the original specification document [1], allow to write the number of possible choices for Q as $\sharp Q = \left[\prod_{i \in \{m_0,\dots,m_{n_0-1}\}} \binom{p}{i}\right]^{n_0}$.

We also consider the possibility that an attacker performs an exhaustive key search employing a quantum computer. In such a case, the best scenario for the attacker is that it is possible to exploit Grover's algorithm to compute and test the value of the public key. Assuming conservatively that the test can be implemented, 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.

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 which is not found via exhaustive search 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 information to speed up a key recovery will in turn have a computational complexity matching or exceeding the required security level.

We note that, for all parameter sets proposed in the original specification [1], the cost of enumerating H and Q exceeds that of the best attacks via ISD.

3.2 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 information set decoding (ISD) algorithms. Such algorithms have a long development history, dating back to the early '60s [26], 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 [4]). 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 allows 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 [26], Lee and Brickell [17], Leon [18], Stern [28], Finiasz and Sendrier [9], and Becker, Joux, May and Meurer (BJMM) [4] 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 [6], which are the same employed in the original specification [1]. Since complete and detailed formulas are available only for the ISD algorithms proposed by Lee and Brickell, and Stern [28], 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 [19] is faster than a quantum version of Stern's [15], 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 [27]. 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 representation $H' = HQ$ in order to attack the LEDA cryptosystems is trying to recover a codeword of the code described by H'. Indeed, 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.

Any sparse row of H' is a low-weight codeword belonging to the dual of the public code. Therefore, it is possible to re-purpose the ISD procedures to perform such a codeword retrieval more efficiently 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 Parameter design procedure

In this section we describe an automated procedure for the design of tight and optimal sets of parameters for the LEDA cryptosystems. This procedure is available in a public domain software implementation at https://github.com/LEDAcrypt/LEDAtools.

Concerning the DFR of the designed system instances, we consider a parameter ϵ that tunes the expected DFR for the code, as described in the specification [1]. The instances originally proposed for the NIST competition have been designed with $\epsilon = 0.3$: the resulting DFR values were in the range 10^{-9}–10^{-8}. In this section, for the sake of comparison, we consider the same DFR target. This is sufficient in those cases were reaction attacks are not relevant, as in LEDAkem with ephemeral keys and indistinguishability under chosen plaintext attack (IND-CPA). When a negligible DFR is required (e.g., for guaranteeing a very long life of any keypair or as an assumption for proving indistinguishability under chosen ciphertext attack (IND-CCA)), the approach described in Section 6 can be followed.

The LEDA cryptosystems design procedure described in this section takes as input the desired security level λ_c and λ_q, expressed as the base-2 logarithm of the number of operations of the desired computational effort on a classical and quantum computer, respectively. In addition to λ_c and λ_q, the procedure also takes as input the number of circulant blocks, $n_0 \in \{2, 3, 4\}$, forming the parity-check matrix H, allowing tuning of the code rate. As a third and last parameter, the procedure takes as input the value of ϵ, which tunes the system DFR. The parameter design procedure outputs the size of the circulant blocks, p, the weight of a column of H, d_v, the number of intentional errors, t, the weights of the n_0 blocks of a row of Q, i.e., $\langle m_0, m_1, \dots, m_{n_0-1}\rangle$, with $\sum_{i=0}^{n_0-1} m_i = m$. The procedure enforces the following constraints on the parameter choice:

– Classical and quantum exhaustive searches for the values of H or Q should require at least 2^{λ_c} and 2^{λ_q} operations. This constraint binds the value of the circulant block p and the weight of a row of the circulant block, d_v for H and, m_i for Q, to be large enough.
– The minimum cost for a message recovery via ISD on both quantum and classical computers must exceed 2^{λ_q} and 2^{λ_c} operations, respectively. This constraint binds the values of the code length $n = n_0 p$, the code dimension $k = (n_0 - 1)p$ and the number of errors t to be chosen such that an ISD on the code $\mathcal{C}(n, k, t)$ requires more than 2^{λ_q} or 2^{λ_c} operations on a quantum and classical computer.
– The minimum cost for a key recovery attack via ISD on both quantum and classical computers must exceed 2^{λ_q} and 2^{λ_c} operations, respectively. This constraint binds the values of the code length $n = n_0 p$, the code redundancy $r = p$ and the number of ones in a row of HQ, $d'_v n_0$, with $d'_v = d'_v m$ to be

chosen such that an ISD on the code $\mathcal{C}(n, r, d'_v n_0)$ requires more than 2^{λ_q} or 2^{λ_c} operations on a quantum and classical computer.

- The choice of the circulant block size, p, should be such that p is a prime number and such that $\mathrm{ord}_2(p) = p - 1$ [1].
- 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 [1].
- The weights of the circulant blocks of Q must guarantee the existence of its multiplicative inverse according to the criterion defined in the LEDA specification [1], i.e., the permanent of the matrix of the block weights must be odd.

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

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

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

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

[3] Note that, in the pseudocode of Algorithm 1, 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 1: LEDAkem/LEDApkc Parameter Generation

```
Input: $\lambda_c, \lambda_q$:desired security levels against classical and quantum attacks, respectively;
       $\epsilon$: 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      $p \leftarrow$ NextPrime($p$)
4      $n \leftarrow n_0 p$, $k \leftarrow (n_0-1)p$, $r \leftarrow p$
5      $t \leftarrow 1$
6      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          $t \leftarrow t+1$
8      $d'_v \leftarrow 4$
9      repeat
10         $d_v \leftarrow \lfloor\sqrt{d'_v}\rfloor - 1 - (\lfloor\sqrt{d'_v}\rfloor \bmod 2)$
11         repeat
12             $d_v \leftarrow d_v + 2$
13             $m \leftarrow \left\lceil \frac{d'_v}{d_v} \right\rceil$
14             $\langle m_0, m_1, \cdots, m_{n_0-1}\rangle$, PermanentOk $\leftarrow$ FindmPartition($m, n_0$)
15         until PermanentOk = true $\vee$ $(m < n_0)$
16         if $(m > n_0)$ then
17             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             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         else
20             SecureOk $\leftarrow$ false
21         $d'_v \leftarrow d'_v + 1$
22     until (SecureOk = true $\vee\ d'_v n_0 \ge p$)
23     if (SecureOk = true) then
24         $p_{th} \leftarrow \mathtt{BF_{th}}(n_0, m d_v, t)$
25     else
26         $p_{th} \leftarrow p$
27 until $p > p_{th}(1+\epsilon)$
28 return $(p, t, d_v, m, \langle m_0, m_1, \cdots, m_{n_0-1}\rangle)$
```

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

Algorithm 1 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 [1] (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 1 is executed through picking a larger value of p and new values for the remaining parameters.

We note that, while this procedure provides a sensible estimation of the fact that the QC-LDPC code employing the generated parameters will correct the computed amount of errors, this is no substitute for a practical DFR evaluation, which is then performed through Montecarlo simulations. Alternatively, the approach described in Section 6 for achieving guaranteed DFR can be followed. Willing to target a DFR of 10^{-9}, we enlarged heuristically the value of p until the target DFR was reached, (in steps of 5% of the value found by the tool). The enlargement took place for: (Category 1, $n_0 = 4$, once).
The C++ tool provided follows the computation logic described in Algorithm 1, but 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 precomputed up to 119981 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 the algorithm starts scanning the list of primes linearly from $\bar{p}$ backwards to find the smallest prime which satisfies the constraints.

The C++ tool relies on Victor Shoup's NTL library (available at `https://www.shoup.net/ntl/`), in particular for the arbitrary precision integer computations

Table 2. Parameter sizes obtained with the parameter design tool, compared to the ones appearing in the original specification [1]

		Revised					Original Submission				
NIST Cat.	n_0	p	t	d_v	m	errors out of decodes	p	t	d_v	m	errors out of decodes
1	2	$15,013$	143	9	$[5,4]$	0 out of $1 \cdot 10^9$	$27,779$	224	17	$[4,3]$	19 out of $2.22 \cdot 10^9$
	3	$9,643$	90	13	$[3,2,2]$	1 out of $1 \cdot 10^9$	$18,701$	141	19	$[3,2,2]$	0 out of $1 \cdot 10^9$
	4	$8,467$	72	11	$[3,2,2,2]$	0 out of $1 \cdot 10^9$	$17,027$	112	21	$[4,1,1,1]$	0 out of $1 \cdot 10^9$
3	2	$24,533$	208	13	$[5,4]$	0 out of $1 \cdot 10^8$	$57,557$	349	17	$[6,5]$	0 out of $1 \cdot 10^8$
	3	$17,827$	129	15	$[4,3,2]$	0 out of $1 \cdot 10^8$	$41,507$	220	19	$[3,4,4]$	0 out of $1 \cdot 10^8$
	4	$14,717$	104	15	$[3,2,2,2]$	0 out of $1 \cdot 10^8$	$35,027$	175	17	$[4,3,3,3]$	0 out of $1 \cdot 10^8$
5	2	$37,619$	272	11	$[7,6]$	0 out of $1 \cdot 10^8$	$99,053$	474	19	$[7,6]$	0 out of $1 \cdot 10^8$
	3	$28,477$	172	13	$[5,4,4]$	0 out of $1 \cdot 10^8$	$72,019$	301	19	$[7,4,4]$	0 out of $1 \cdot 10^8$
	4	$22,853$	135	13	$[4,3,3,3]$	0 out of $1 \cdot 10^8$	$60,509$	239	23	$[4,3,3,3]$	0 out of $1 \cdot 10^8$

and the tunable precision floating point computations and requires a compiler supporting the C++11 standard.

5 Revised parameter sets and figures of merit

In this section we report a comparison of the parameters of the LEDA cryptosystems obtained with the design procedure described in Section 5, which are reported in Table 2, in comparison with those reported in the original specification [1]. Deriving the parameters in Table 2 took approximately a day for all the parameter sets with $n_0 \in \{3,4\}$ and approximately a month for all the parameters 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.

As shown in Table 2, targeting the same computational effort of breaking AES on a classical and quantum computer yields parameters sets having a code of roughly half the length of the current LEDA cryptosystems. Such a fact is justified by the former parameters targeting a higher security level, i.e., 2^{128}, 2^{192}, 2^{256} operations on a quantum computer. The revised parameter sets allow to achieve the same DFR (experimentally validated via Montecarlo experiments) as the ones in the submission documents.

5.1 Resulting computational complexities of the 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 3.1 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

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) Classical	Quantum	Q enumeration cost ($\log_2$ ♯quantum gates) Classical	Quantum
1	2	212.78	106.39	226.74	113.37
	3	418.53	209.26	264.18	132.09
	4	473.05	236.52	447.37	223.68
3	2	314.06	157.03	239.49	119.74
	3	514.70	257.35	356.77	178.38
	4	669.67	334.83	476.08	238.04
5	2	283.87	141.93	351.59	175.79
	3	479.48	239.74	528.86	264.43
	4	622.80	311.40	703.60	351.80

a classical computer. From the results in Table 3 it is straightforward to note that, despite the reduction in key sizes from the ones proposed in the submission document, an exhaustive search on either H or Q is still clearly above the required computational effort.

Then, as described in Section 3.2, the two main attacks that can be mounted against the considered systems are message recovery attacks and key recovery attacks based on ISD algorithms. The complexity of these attacks against the new system instances is reported in Table 4 and Table 5. An interesting point to be noted is that, while providing clear asymptotic speedups, the improvements to the ISD algorithms proposed since Stern's [28] 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 LEDA cryptosystems paramteters. Concerning quantum ISDs, it is interesting to notice that the quantum variant of the Stern algorithm as described by de Vries [6] does not achieve an effective speedup when compared against a quantum transposition of Lee and Brickell's ISD. Such a result can be ascribed to the fact that the speedup obtained by the reduction in the number of ISD iterations which can be obtained by Stern's ISD is mitigated by the fact that the overall number of iterations to be run is quadratically reduced by applying Grover's algorithm to execute them.
Comparing the computational complexities of the message decoding (Table 4) and the key recovery attack (Table 5), we note that performing a message recovery attack is always easier than the corresponding key recovery attack on the same parameter set, albeit by a small margin.

5.2 Performance of the cryptosystems with revised parameters

Willing to provide a preliminary gauge of the performance and keysize improvements obtained from the new parameter sets proposed in this document, we

Table 4. Cost of performing a message recovery attack, i.e., an ISD on the code $\mathcal{C}(n_0p, (n_0-1)p, t)$, for the values of 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 [26]	L-B [17]	Leon [18]	Stern [28]	F-S [9]	BJMM [4]	Q-LB [6]	Q-Stern [6]
1	2	176.12	165.14	163.29	149.83	149.81	146.40	100.73	102.13
	3	174.76	164.27	161.33	148.84	148.83	145.46	99.96	101.37
	4	176.03	165.68	162.10	149.86	149.85	147.43	100.82	101.57
3	2	243.03	230.98	229.09	213.51	213.50	210.85	134.53	135.94
	3	238.83	227.31	224.25	209.74	209.73	207.22	132.59	133.99
	4	242.09	230.68	227.00	212.69	212.68	210.13	134.31	135.71
5	2	308.65	295.82	293.87	276.78	276.78	274.51	167.73	169.13
	3	308.72	296.37	293.25	277.12	277.11	274.21	167.97	169.37
	4	305.71	293.55	289.79	274.01	274.01	271.22	166.54	167.94

Table 5. Cost of performing a key recovery attack, i.e., an ISD on the code $\mathcal{C}(n_0p, p, n_0d_vm)$, for the 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 [26]	L-B [17]	Leon [18]	Stern [28]	F-S [9]	BJMM [4]	Q-LB [6]	Q-Stern [6]
1	2	188.32	176.98	175.19	161.01	161.00	157.97	103.18	104.59
	3	187.53	175.62	174.16	159.53	159.50	156.14	103.14	104.55
	4	193.53	181.22	179.00	164.77	164.74	161.70	106.59	108.00
3	2	261.91	249.51	247.68	231.42	231.41	228.83	140.16	141.56
	3	266.69	253.65	251.58	235.24	235.22	232.66	143.05	144.45
	4	254.97	241.77	238.99	223.43	223.41	220.05	137.66	139.07
5	2	315.12	302.15	300.22	282.85	282.84	280.60	167.09	168.49
	3	327.69	314.00	311.52	294.21	294.21	291.19	173.90	175.30
	4	312.69	298.86	295.66	279.13	279.12	276.06	166.84	168.24

report the results of the execution of both LEDAkem and LEDApkc employing the parameter sets reported in Table 2. We report, for the sake of a clear comparison, also the execution time of the LEDAkem and LEDApkc employing the original submission parameters on the same host. As shown in Table 6 and Table 8 the running times of the LEDAkem and LEDApkc instances employing the optimized parameter sets achieve a speedup in between 3× and 6.5× thanks to the reduction in both the code length and in the number of non null terms in both H and Q. The speedups are the result of a roughly 2× reduction in the size of the code instances, and a consequent reduction in the number of non null terms in H and Q. Such speedups are coherent with the quadratic complexity of the reference implementation of the polynomial multiplication employed in the encryption phase, and the $\mathcal{O}((d_v + m)n)$ complexity of the decoder. It is worth noting that a complete ephemeral KEM, with the architecture independent reference implementation completes in 10 to 15 milliseconds for NIST Category 1 parameters.

Table 7 and Table 9 reports the keypair sizes for LEDAkem and LEDApkc, respectively, together with the size of the encapsulated shared secret and ciphertext. The parameter optimization presented in this official comment allows LEDAkem to have public key sizes which never exceed 10 kB for all the NIST categories, and are below 3.5 kB for Category 1. It is also noteworthy that the encapsulated secret for Category 1 are below 1500 kB for both $n_0 = 3$ and $n_0 = 4$, allowing them to fit into the payload of a single Ethernet frame, with enough room to contain the TCP/IP headers. LEDApkc allows plaintext sizes ranging between 2 kB and 8.6 kB, depending on the NIST category, allowing to encrypt directly a reasonable amount of information (e.g., some pages of an ASCII text message).

Table 6. Running times for key generation, encryption and decryption of LEDAkem as a function of the chosen category and number of circulant blocks n_0 on an Intel Skylake i5-6600 at 3.6 GHz. The figures are taken employing the completely portable reference implementation in ISO C99, compiled with `GCC` 6.3.0, employing `-march=native -O3` as optimization parameters

Set	Category	n_0	KeyGen (ms)	Encrypt (ms)	Decrypt (ms)	Ephemeral KEM (ms)
Revised	1	2	10.90 (± 0.35)	0.56 (± 0.03)	4.31 (± 0.49)	15.77
		3	4.91 (± 0.22)	0.52 (± 0.03)	5.69 (± 0.33)	11.14
		4	3.76 (± 0.17)	0.62 (± 0.04)	6.31 (± 0.62)	10.70
	2–3	2	28.64 (± 0.48)	1.40 (± 0.08)	12.56 (± 1.19)	42.61
		3	15.59 (± 0.39)	1.56 (± 0.08)	13.91 (± 0.87)	31.07
		4	10.98 (± 0.34)	1.52 (± 0.10)	17.37 (± 0.75)	29.88
	4–5	2	67.20 (± 1.21)	2.92 (± 0.10)	16.98 (± 0.66)	87.10
		3	39.41 (± 0.63)	3.49 (± 0.17)	23.72 (± 1.44)	66.63
		4	26.16 (± 0.75)	3.48 (± 0.09)	24.43 (± 1.05)	54.07
Submission	1	2	38.62 (± 0.71)	1.76 (± 0.13)	18.10 (± 0.54)	58.48
		3	17.58 (± 0.41)	1.77 (± 0.13)	21.72 (± 1.66)	41.08
		4	15.16 (± 0.41)	2.17 (± 0.15)	24.75 (± 1.49)	42.09
	2–3	2	161.91 (± 1.63)	6.71 (± 0.27)	50.30 (± 4.14)	218.93
		3	85.89 (± 0.70)	7.42 (± 0.30)	50.33 (± 1.84)	143.65
		4	62.33 (± 0.84)	7.94 (± 0.30)	49.45 (± 2.09)	119.73
	4–5	2	514.19 (± 2.94)	18.85 (± 0.39)	95.71 (± 4.67)	628.76
		3	257.65 (± 1.67)	20.32 (± 0.40)	95.16 (± 2.23)	373.13
		4	182.91 (± 2.66)	21.98 (± 0.47)	135.67 (± 4.22)	340.57

Table 7. Sizes of the key pair and encapsulated shared secret for LEDAkem as a function of the chosen category and number of circulant blocks n_0

Set	Category	n_0	Private Key (B) At rest	Private Key (B) In memory	Public Key (B)	Encap. secret (B)	Shared secret (B)
Revised	1	2	24	468	1,880	1,880	32
		3	24	604	2,416	1,208	32
		4	24	716	3,192	1,064	32
	2–3	2	32	644	3,072	3,072	48
		3	32	828	4,464	2,232	48
		4	32	924	5,520	1,840	48
	4–5	2	40	764	4,704	4,704	64
		3	40	988	7,120	3,560	64
		4	40	1,092	8,592	2,864	64
Submission	1	2	24	668	3,240	3,240	32
		3	24	844	4,688	2,344	32
		4	24	1,036	6,408	2,136	32
	2–3	2	32	972	7,200	7,200	48
		3	32	1,196	10,384	5,192	48
		4	32	1,364	13,152	4,384	48
	4–5	2	40	1,244	12,384	12,384	64
		3	40	1,548	18,016	9,008	64
		4	40	1,772	22,704	7,568	64

Table 8. Running times for LEDApkc primitives as a function of the chosen category and number of circulant blocks n_0 on an Intel Skylake i5-6600 at 3.6 GHz. The figures are taken employing the completely portable reference implementation in ISO C99, compiled with `GCC` 6.3.0, employing `-march=native -O3` as optimization parameters

Set	Category	n_0	KeyGen (ms)	Encrypt (ms)	Decrypt (ms)
Revised	1	2	13.07 (± 0.37)	0.75 (± 0.05)	4.77 (± 0.51)
		3	5.75 (± 0.23)	0.75 (± 0.04)	6.04 (± 0.40)
		4	4.63 (± 0.16)	0.94 (± 0.08)	6.54 (± 0.62)
	2–3	2	33.99 (± 0.65)	1.60 (± 0.08)	13.42 (± 1.03)
		3	18.46 (± 0.28)	1.94 (± 0.12)	14.90 (± 0.71)
		4	13.01 (± 0.33)	2.15 (± 0.15)	18.22 (± 0.83)
	4–5	2	79.36 (± 1.45)	3.34 (± 0.18)	18.51 (± 0.89)
		3	46.72 (± 0.95)	4.20 (± 0.22)	25.20 (± 0.98)
		4	30.62 (± 0.58)	4.35 (± 0.14)	26.46 (± 1.27)
Submission	1	2	45.30 (± 1.69)	3.11 (± 0.06)	20.87 (± 0.65)
		3	20.96 (± 0.23)	3.10 (± 0.06)	25.18 (± 2.18)
		4	17.99 (± 0.22)	3.94 (± 0.08)	28.30 (± 0.80)
	2–3	2	198.49 (± 1.41)	12.06 (± 0.18)	62.55 (± 4.57)
		3	100.39 (± 0.57)	13.06 (± 0.15)	57.58 (± 2.69)
		4	72.78 (± 0.31)	14.18 (± 0.22)	59.75 (± 1.91)
	4–5	2	558.84 (± 3.41)	33.96 (± 0.21)	115.36 (± 4.08)
		3	298.91 (± 4.18)	37.28 (± 0.61)	116.93 (± 5.07)
		4	208.90 (± 0.71)	39.85 (± 0.25)	157.23 (± 4.18)

Table 9. Sizes of the key pair, plaintext and ciphertext for LEDApkc as a function of the chosen category and number of circulant blocks n_0

Set	Category	n_0	Private Key Size (B)		Public Key	Max Plaintext	Ciphertext
			At rest	In memory	size (B)	size (B)	size (B)
Revised	1	2	24	468	1,880	2,001	3,760
		3	24	604	2,416	2,483	3,624
		4	24	716	3,192	3,231	4,256
	2–3	2	32	644	3,072	3,251	6,144
		3	32	828	4,464	4,565	6,696
		4	32	924	5,520	5,602	7,360
	4–5	2	40	764	4,704	4,950	9,408
		3	40	988	7,120	7,269	10,680
		4	40	1,092	8,592	8,681	11,456
Submission	1	2	24	668	3,480	3,690	6,960
		3	24	844	4,688	4,813	7,032
		4	24	1,036	6,408	6,496	8,544
	2–3	2	32	972	7,200	7,558	14,400
		3	32	1,196	10,384	10,608	15,576
		4	32	1,364	13,152	13,320	17,536
	4–5	2	40	1,244	12,384	12,897	24,768
		3	40	1,548	18,016	18,336	27,024
		4	40	1,772	22,704	22,955	30,272

6 Bounded DFR for Q-decoders

Binary block error correction codes $\mathcal{C}(n,k)$ 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 as *syndrome*, $e \in \mathbb{Z}_2^n$ is a $1 \times n$ binary vector with a given number $t \ll n$ of non-null entries named as error vector, representing the unknown sequence of values to be found, while H' is assumed to have $d_v \ll n$ non-null entries per column. Therefore, the purpose of an iterative decoding procedure is to compute the values of the elements of e given H', s.

A common approach to perform iterative decoding is the *Bit Flipping* (BF) strategy firstly described in [11]. Such an approach considers the i-th row of H', with $i \in \{0, \dots, r-1\}$, as a representation of the coefficients of the parity check equation involving the unknown e_j, with $j \in \{0, \dots, n-1\}$, 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., $\hat{e} = 0_{1\times n}$ (which means that 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 analyses the parity check equations in which a given error variable $\hat{e}_j$ is involved, with $j \in \{0, \dots, n-1\}$, i.e., the number of rows of H' where the j-th element is non-null, and counts how many of them are unsatisfied, i.e., counts how many equations where there is a contribution of the unknown e_j have a constant term in the syndrome which is non-null. Such a count of the number of unsatisfied parity check equations, $\texttt{upc}_j$, 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 an integer vector ($\varsigma \leftarrow \texttt{Lift}(s)$) by an integer matrix ($\mathcal{H}' \leftarrow \texttt{Lift}(H')$) product obtaining a $1 \times n$ integer vector $\texttt{upc}^{(\mathrm{BF})}$, i.e., $\texttt{upc}^{(\mathrm{BF})} \leftarrow \varsigma\, \mathcal{H}'$.

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 $\texttt{upc}_j^{(\mathrm{BF})}$ 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 an alternate decoding strategy, which, while retaining a fix point bit-flipping approach, is more efficient than the garden variety bit-flipping which was described. Such a procedure, known as the *Q-decoder*, relies on the fact that the (secret) parity check matrix of the LEDA cryptosystems H' is obtained as the product of two low-density matrices, i.e., $H' = HQ$, where H has size $r \times n$ and number of non-null elements in a row equal to

$d_c \ll n$, while Q has size $n \times n$ and number of non-null elements in a row equal to $m \ll n$. Due to the sparsity of both H and Q, their product HQ has a number of non-null row elements $\leq d_c m$, with the equality sign 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 $\mathcal{H} \leftarrow \texttt{Lift}(H)$ and $\mathcal{Q} \leftarrow \texttt{Lift}(Q)$, 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 $\texttt{upc}^{(\mathrm{temp})}$ is computed in the same fashion as in the BF decoder, employing the lifted syndrome, $\varsigma \leftarrow \texttt{Lift}(s)$, and $\mathcal{H}$ instead of $\mathcal{H}'$, i.e., $\texttt{upc}^{(\mathrm{temp})} \leftarrow \varsigma\, \mathcal{H}$. The value of the actual $1 \times n$ integer vector $\texttt{upc}^{(\mathrm{Q-dec})}$ storing the unsatisfied parity-check counts is then computed as: $\texttt{upc}^{(\mathrm{Q-dec})} \leftarrow \texttt{upc}^{(\mathrm{temp})}\ \mathcal{Q}$.

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 $\hat{e}_j$, $j \in \{0, \dots, n-1\}$, for which the j-th unsatisfied parity-check count $\texttt{upc}_j^{(\mathrm{Q-dec})}$ exceeds the chosen threshold b. Subsequently, the value of the syndrome s is recomputed 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 [1]).

The Q-decoder terminates with success its decoding procedure if s is null or with a decoding failure if s is not null but the allowed number of iterations $l_{\max}$ is exceeded.

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 $\mathcal{H}' \leftarrow \texttt{Lift}(H')$, $\mathcal{H} \leftarrow \texttt{Lift}(H)$, $\mathcal{Q} \leftarrow \texttt{Lift}(Q)$, the matrices obtained through lifting their values in the integer domain. 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 $\mathcal{H}' = \mathcal{H}\mathcal{Q}$, the BF and Q-decoding procedures compute as output same values for s, and $\hat{e}$.*

Proof. The functional equivalence can be proven showing that the update to the two state vectors, the syndrome s and the current $\hat{e}$ performed by the bit-flipping decoder and the Q-decoder leads to the same values at the end of each iteration of the decoding algorithms. We start by observing that the second phase of the BF and Q-decoder procedure will lead to the same state update of s and $\hat{e}$ if the values of the $\texttt{upc}^{(\mathrm{BF})}$ vector for the BF procedure and the $\texttt{upc}^{(\mathrm{Q-dec})}$ vector for the Q-decoder coincide. Indeed, since the update only depends on the values of the unsatisfied parity-checks and the flipping threshold

b, if $\mathtt{upc}^{(\mathrm{BF})} = \mathtt{upc}^{(\mathrm{Q-dec})}$ 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 $\mathtt{upc}^{(\mathrm{BF})}_j = \sum_{z=0}^{r-1} h'_{zj}\, s_z$. 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 $\mathtt{upc}^{(\mathrm{Q-dec})} = \mathtt{upc}^{(\mathrm{BF})}$. □

We point out that, when $\mathcal{H}' \neq \mathcal{H}\mathcal{Q}$, it is not possible to state the equivalence of the two procedures; however, some considerations about their behaviour can be drawn as follows. The entry in the i-th row, j-th column of $\mathcal{H}'$ is different from the one with the same coordinates in $\mathcal{H}\mathcal{Q}$ when the scalar product between the i-th row of $\mathcal{H}$ and the j-th column of $\mathcal{Q}$ equals an even number. The probability with which such an event occur can be quantified as $\sum_{i=2}^{\min\{m,d_c\}} \frac{\binom{m}{i}\binom{n-m}{d_c-i}}{\binom{n}{d_c}}$. It is worth noting that such a probability becomes negligible when the code parameters (n,d_c,m) take values of practical interest. Therefore, we have that either $\mathcal{H}' = \mathcal{H}\mathcal{Q}$ or the two matrices differ in a few number of entries implying that the two decoding procedures will both return, with overwhelming probability, the same result as differences in a few number entries in $\mathcal{H}'$ and $\mathcal{H}\mathcal{Q}$ can cause only a few differences in the computation of the values of the unsatisfied parity check counts $\mathtt{upc}^{(\mathrm{BF})}$ and $\mathtt{upc}^{(\mathrm{Q-dec})}$.

For the sake of simplicity, in the following, we assume that previous Lemma holds and omit the superscript for denoting the vector of the unsatisfied parity-check count $\mathtt{upc}$.

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-null entries of a column of H is $d_v \ll n$, the number of non-null entries of a column of Q is $m \ll n$, and the number of non-null entries of a column of 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. $\square$

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 bit-flipping threshold b should be lower than the value of the unsatisfied parity-check count $\texttt{upc}_j$, $j \in \{0, \dots, n-1\}$ related to a value for which the guessed value of the j-th error bit, $\hat{e}_j$, in the current iteration, is different from the actual (and unknown) error vector j-th bit value, e_j (i.e., $e_j \neq \hat{e}_j$) so that such $\hat{e}_j$ is rightfully flipped.
(ii) The bit-flipping threshold b should be higher than an unsatisfied parity-check count $\texttt{upc}_j$ related to a value for which $e_j = \hat{e}_j$ so that, in the current iteration, the bit value in the guessed error vector $\hat{e}_j$ is rightfully not flipped.

Observing that during the decoding procedure the j-th bit value of the guessed error vector $\hat{e}$ is flipped when $\texttt{upc}_j$ is higher than or equal to b, an ideal case where it is possible to attain a null decoding failure rate (DFR) is the one in which, whenever the maximum possible value of any unsatisfied parity-check count $\texttt{upc}_j$ related to a variable $\hat{e}_j = e_j$ (i.e., no flip is needed) is lower than the minimum possible value of $\texttt{upc}_k$ 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 $\texttt{max\_upc}_{no\,flip} < b \leq \texttt{min\_upc}_{flip}$ 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 analyse the contribution to the values of $\texttt{upc}$ provided by the bits of the actual error vector. Let $u_z \in \mathbb{Z}_2^n$, with $z \in \{0, \dots, n-1\}$, denote $1 \times n$ binary vectors such that only the z-th component of any u_z is not null (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 $\mathbf{U} = \{u \in \mathbb{Z}_2^n,\ wt(u) = 1, z \in \mathbf{I}\}$, where $\mathbf{I} \subset \{0, \dots, n-1\}$ and $|\mathbf{I}| = t$ (thus it also holds $|\mathbf{U}| = t$), and quantify the contributions of each bit in e to the value of $\texttt{upc}_z$ computed by the Q-decoder in its first iteration, proceeding backwards from each $u \in \mathbf{U}$ 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 $\mathcal{C}(5, 3)$, assuming that a single bit in the actual error vector is asserted, i.e., $e = u_2$. Figure 1 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 $\texttt{upc}_z$, $z \in \{0, \dots, n-1\}$[4].

[4] Note that the notation $P_{(z)}$ denote a matrix whose values are related with the bit in position z of the actual (unknown) error vector (it may include repeated rows).

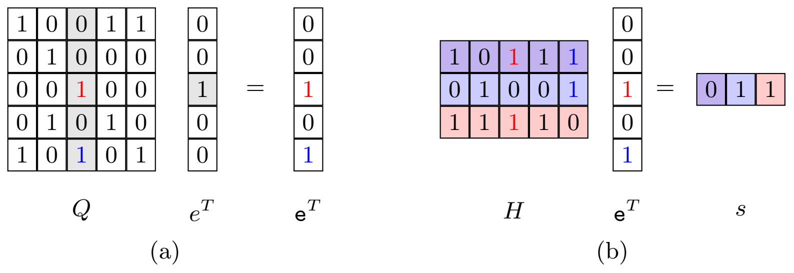

Fig. 1. Steps of the syndrome computation process of a toy code $\mathcal{C}(5,3)$, having H with constant column weight $d_v = 2$ and Q with constant column weight $m = 2$. The single bit error vector employed is $e = u_2 = (0,0,1,0,0)$. In (a) the error vector is expanded as $\mathtt{e} = u_2 + u_4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

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 $z \in \{0,\dots,n-1\}$, (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., $s = H\left(\left(\mathtt{e}^{(z)}\right)^T\right)^T$.

Consider $\mathtt{e}^{(z)}$ as the sum of m binary vectors u_j, $j \in \{0,\dots,n-1\}$ with a single non-null entry in the j-th position (see $\mathtt{e}^{(2)} = u_2 + u_4$, with u_2 highlighted in red and u_4 in blue in Fig. 1). Each u_j in $\mathtt{e}^{(z)}$ is involved in all the parity-check equations of H, i.e., the $1 \times n$ rows, H_i, with $i \in \{0,\dots,r-1\}$, 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 $\mathtt{e}^{(z)}$. The $P_{(z)}$ matrix allows to compute the contribution to the value $\mathtt{upc}_z$, $z \in \{0,\dots,n-1\}$ provided by any expanded error vector $\mathtt{e}^{(j)}$, $j \in \{0,\dots,n-1\}$, 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, $z \in \{0,\dots,n-1\}$, 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 $\mathtt{upc}_z \leftarrow wt\left(P_{(z)}\left(\mathtt{e}^{(z)}\right)^T\right)$.

The construction of $P_{(z)}$ for the toy example is reported in Fig. 1, 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_1 and H_2 as they both involve u_4.

Figure 2, 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. 1. Indeed, computing the value of $\mathtt{upc}_2$ as

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.

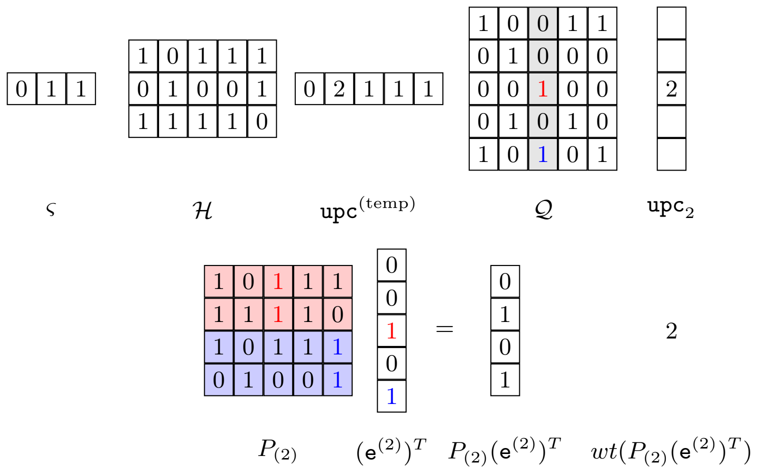
ς
$\mathcal{H}$
$\texttt{upc}^{(\text{temp})}$
$\mathcal{Q}$
$\texttt{upc}_2$
$P_{(2)}$
$(\texttt{e}^{(2)})^T$
$P_{(2)}(\texttt{e}^{(2)})^T$
$wt(P_{(2)}(\texttt{e}^{(2)})^T)$
=
2

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

the third component of the vector $\varsigma\mathcal{H}\mathcal{Q}$ yields the same results as computing the binary vector $P_{(2)}(\texttt{e}^{(2)})^T$ and computing its weight.

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

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

6.1 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 $\texttt{upc}_z$, following the work in [29] 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 $\texttt{upc}_z$ with $z \in \{0, \dots, n-1\}$ such that $e_z = 1$, match or exceed the threshold b, and all the $\texttt{upc}_z$ 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 $e = \sum_{i \in \mathbf{I}} u_i$, $\mathbf{I} \subset \{0, \dots, n-1\}, |\mathbf{I}| = t$ 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*

$$\alpha = \min_{z \in \mathbf{I}} \left\{ wt\left(P_{(z)} \left(\mathtt{e}^{(z)}\right)^T \right) \right\}, \quad \beta = \max_{\substack{z,i \in \mathbf{I} \\ z \neq i}} \left\{ wt\left(P_{(z)} \left(\mathtt{e}^{(z)}\right)^T \wedge P_{(z)} \left(\mathtt{e}^{(i)}\right)^T \right) \right\}$$

$$\gamma = \max_{\substack{z,i \in \mathbf{I} \\ z \neq i}} \left\{ wt\left(P_{(z)} \left(\mathtt{e}^{(i)}\right)^T \right) \right\}.$$

Proof. We first determine the lower bound $\mathtt{min\_upc}_{flip}$ as the lowest value of an unsatisfied parity-check count $\mathtt{upc}_z$, $z \in \{0, \dots, n-1\}$, 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 $i \in \{0, \dots, n-1\}$ and $wt(u_i) = 1$, i.e., $e = \sum_{i \in \mathbf{I}} u_i$, $\mathbf{I} \subset \{0, \dots, n-1\}, |\mathbf{I}| = t$, keeping into account that $z \in \mathbf{I}$, i.e., u_z contribute to the error vector e as 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 need to be flipped when it is right to do so.

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

Considering that, for a generic pair of binary vectors a, b of length n we have that $wt(a \oplus b) = wt(a) + wt(b) - 2wt(a \wedge b)$, where $\wedge$ indicates the component-wise binary product (i.e., the Boolean `and`), we expand the former onto

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

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 $\texttt{upc}_z$, we obtain

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

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

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

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, $wt\left(c \wedge (a \oplus b)\right) \leq wt\left(c \wedge (a \vee b)\right) = wt\left((c \wedge a) \vee (c \wedge b)\right)$. Thus, the second added in the previous inequality can be replaced by the following quantity:

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

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

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

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

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

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 $\mathtt{upc}_z$, $z \in \{0,\dots,n-1\}$, 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 $u_z \notin \mathbf{U}$ (and thus $z \notin \mathbf{I}$).

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

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:

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

□

6.2 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 $\{0,\dots,t\}$, $t \geq 0$ having a probability mass function $Pr\left[\mathtt{T} = \tau\right]$, $\tau = wt\left(\hat{e}^* \oplus e\right)$ depending on the decoding strategy and the LDPC code parameters.

To quantify the said probability we consider the decoding procedure employed by the LEDA cryptosystems assuming that Lemma 1 holds. Given the equivalence of the BF decoder and Q-decoder provided by the said 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 as the ones employed by the LEDA cryptosystem thus, $r = (n_0 - 1)p$, $n = n_0 p$, p a prime number, $n_0 \in \{2, 3, 4\}$, while m denotes the number of non-null entries in each row/column of the $n \times n$ matrix Q, and $d_v\, n_0$ denotes the number of non-null 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-null 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 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 it.

It is easy to acknowledge that the said 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:

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

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 it.

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

The probability $\mathtt{p}_{\mathtt{correct}}$ that the $\mathtt{upc}$ 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.:

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

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

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

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

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

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

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 $\tau \in \{0, \dots, t\}$ 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 results allows us to estimate the probability of having a given number of errors $\tau \in \{0, \dots, t\}$ 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.

6.3 Two iterations Q-decoder with constrained DFR

We now combine the results obtained to obtain a closed form upper bound to the DFR of a Q-decoder performing two iterations of the decoding action. To this end, we employ the result derived on the probability that the first iteration

Table 10. Instances of LEDA cryptosystems with security equivalent to NIST category 1 and 3, achieving DFR $< 2^{-64}$ with a two iterations Q-decoder for a significant amount of the randomly selected keypairs.

NIST Category	n_0	p	t	d_v	m	t_{left}	DFR achieving keys out of 100	upper bound on code DFR
1	2	37,501	143	9	$[5, 4]$	4	85 out of 100	$2^{-64.73}$
1	2	36,011	143	7	$[7, 6]$	5	64 out of 100	$2^{-64.55}$
1	2	37,277	143	9	$[6, 5]$	5	67 out of 100	$2^{-64.24}$
1	3	27,011	90	13	$[3, 2, 2]$	3	97 out of 100	$2^{-65.20}$
1	3	25,013	90	11	$[4, 3, 2]$	4	85 out of 100	$2^{-64.51}$
1	3	22,397	90	7	$[5, 4, 4]$	5	44 out of 100	$2^{-64.50}$
1	4	21,523	72	13	$[2, 2, 2, 1]$	3	96 out of 100	$2^{-65.26}$
1	4	20,029	72	11	$[3, 2, 2, 2]$	4	62 out of 100	$2^{-64.89}$
1	4	17,851	72	7	$[4, 3, 3, 3]$	5	30 out of 100	$2^{-64.09}$
3	2	64,013	208	13	$[5, 4]$	4	88 out of 100	$2^{-65.37}$
3	2	60,107	208	7	$[9, 8]$	6	31 out of 100	$2^{-76.10}$
3	3	41,651	129	15	$[4, 3, 2]$	4	4 out of 100	$2^{-64.29}$
3	3	37,307	129	9	$[7, 4, 4]$	6	25 out of 100	$2^{-65.02}$
3	4	33,851	104	15	$[3, 2, 2, 2]$	4	95 out of 100	$2^{-66.55}$
3	4	31,387	104	9	$[4, 4, 4, 3]$	6	14 out of 100	$2^{-76.11}$

of a Q-decoder leaves more than $\bar{t}$ errors to be corrected, combining it with the bound on the number of errors which can be corrected with null DFR.

In particular, given a desired target DFR, p_{dfr}, we design code parameters such that the first iteration of a Q-decoder leaves at most $\bar{t}$ errors with a probability of $1 - p_{dfr}$. For the second iteration we pick the flipping threshold of the Q-decoder as $\bar{t} = \left\lfloor \frac{\alpha+\beta}{\gamma+\beta} \right\rfloor$. As a consequence of Theorem 1, we know that the second iteration will correct all the remaining $\bar{t}$ errors. Noting that the values of α, β and γ depend only on the H and Q, given a chosen target bound for the DFR and a set of parameters for the LEDA cryptosystems, we are able to evaluate the amount of secret keys which allow to reach the desired DFR target. We note that such a procedure can be integrated in the key generation process for LEDApkc, where the concern on the DFR is actually meaningful, as opposed to the ephemeral key based LEDAkem.

We note that the actual DFR is very likely to be lower than the one provided by our bound. Indeed, while there are no guarantees, is it highly likely that the Q-decoder will also correct all the remaining errors during the second iteration in cases where the number exceeds by a small amount the threshold $\bar{t}$.

We report in Table 10 we have shown some possible choices, by imposing a DFR smaller than 2^{-64} and aiming at parameter sets where the probability of drawing a random secret key achieving the desired DFR is significant. To obtain the said parameters, we started from the ones obtained by the automated parameter optimization procedure and enlarged the length of the circulant block, picking a target value for the amount of errors t_{left} which are left not corrected by the first iteration with a probability smaller than 2^{-64}. Once the parameters were selected, we drew 100 keypairs at random for each parameter set, and evaluated how many of them satisfy the inequality $\bar{t} = \frac{\alpha+\beta}{\gamma+\beta}$, thus guaranteeing the correction of the t_{left} errors during the second Q-decoder iteration.

As it can be seen from the results reported in Table 10, the parameter sets we determined are able to achieve a DFR $\leq 2^{-64}$ enlarging the code size by a factor ranging from $2\times$ to $3\times$ with respect to the optimal parameters, while keeping the number of set terms in H and Q equal to the optimal parameter themselves. The obtained LEDA parametrizations show that it is possible to achieve the desired DFR discarding an acceptable number of keypairs, given proper tuning of the parameters. The parameter derivation procedure for bounded DFR code instances can also be automated, and is expected to provide improvements in term of flexibility in choosing a fixed code size or keypair rejection rate and determining optimally the remaining parameters.

References

1. M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini. LEDAkem and LEDApkc website. https://www.ledacrypt.org/.
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, 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.
4. 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.
5. 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.
6. S. de Vries. Achieving 128-bit Security against Quantum Attacks in OpenVPN. Master's thesis, University of Twente, August 2016.
7. 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.
8. T. Fabsic, V. Hromada, and P. Zajac. A reaction attack on LEDApkc. *IACR Cryptology ePrint Archive*, 2018:140, 2018.
9. 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.

10. 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.
11. R. G. Gallager. *Low-Density Parity-Check Codes*. M.I.T. Press, 1963.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. Information Theory*, 34(5):1354–1359, 1988.
19. 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.
20. R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report*, pages 114–116, 1978.
21. 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.
22. 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.

23. 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.
24. National Institute of Standards and Technology. Post-quantum crypto project, Dec. 2016.
25. H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Probl. Contr. and Inf. Theory*, 15:159–166, 1986.
26. E. Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Information Theory*, 8(5):5–9, 1962.
27. 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.
28. 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.
29. 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.
30. 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.