POLITECNICO DI TORINO
Repository ISTITUZIONALE

Efficient hardware implementation of the LEDAcrypt Decoder

Original
Efficient hardware implementation of the LEDAcrypt Decoder / Koleci, K., Santini, P., Baldi, M., Chiaraluce, F., Martina, M., Masera, G.. - In: IEEE ACCESS. - ISSN 2169-3536. - ELETTRONICO. - 9:(2021), pp. 66223-66240. [10.1109/ACCESS.2021.3076245]

Availability:
This version is available at: 11583/2898432 since: 2021-05-26T10:06:48Z

Publisher:
IEEE Access

Published
DOI:10.1109/ACCESS.2021.3076245

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

10 October 2026

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 21, 2021, accepted April 15, 2021, date of publication April 28, 2021, date of current version May 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3076245

Efficient Hardware Implementation of the LEDAcrypt Decoder

KRISTJANE KOLECI[1], (Graduate Student Member, IEEE), PAOLO SANTINI[2], (Member, IEEE), MARCO BALDI[2], (Senior Member, IEEE), FRANCO CHIARALUCE[2], (Senior Member, IEEE), MAURIZIO MARTINA[1], (Senior Member, IEEE), AND GUIDO MASERA[1], (Senior Member, IEEE)

[1]Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Torino, Italy
[2]Department of Information Engineering, Università Politecnica delle Marche, 60121 Ancona, Italy

Corresponding author: Kristjane Koleci (kristjane.koleci@polito.it)

ABSTRACT This work describes an efficient implementation of the iterative decoder that is the main part of the decryption stage in the LEDAcrypt cryptosystem, recently proposed for post-quantum cryptography based on low-density parity-check (LDPC) codes. The implementation we present exploits the structure of the variables in order to accelerate the decoding process while keeping the area bounded. In particular, our focus is on the design of an efficient multiplier, the latter being a fundamental component also in view of considering different values of the cryptosystem's parameters, as it might be required in future applications. We aim to provide an architecture suitable for low cost implementation on both Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) implementations. As for the FPGA, the total execution time is 0.6 ms on the Artix-7 200 platform, employing at most 30% of the total available memory, 15% of the total available Look-up Tables and 3% of the Flip-Flops. The ASIC synthesis has been performed for both STM FDSOI 28 nm and UMC CMOS 65 nm technologies. After logic synthesis with the STM FDSOI 28 nm, the proposed decoder achieves a total latency of 0.15 ms and an area occupation of 0.09 mm^2. The post-Place&Route implementation results for the UMC 65 nm show a total execution time of 0.3 ms, with an area occupation of 0.42 mm^2 and a power consumption of at most 10.5 mW.

INDEX TERMS Applied cryptography, post-quantum cryptography, hardware design, ASIC, FPGA, bit-flipping decoding, LDPC codes.

I. INTRODUCTION

Quantum computing is becoming a reality, besides being an active and appealing research field, due to its rapid advancement in recent years [1]–[4]. The expected computing power of quantum computers can deeply change our world. Quantum computers will enable dramatic reductions in the complexity of solving some widespread problems, but also pose a serious threat on the security of Public Key Cryptography (PKC). One of the security requirements upon which an asymmetric cryptosystem is built is the hardness of discovering the Secret Key (SK) from the Public Key (PK): the PK is computed by applying a one-way function to the SK, and inverting this function should be computationally infeasible. One of the most widespread one-way functions used in current PKC is based on integer factorization. Factorizing large integers is believed to be a Non-Polynomial (NP) problem with classical computers. However, it has been demonstrated that by using Shor's quantum algorithm [5], the problem can actually be solved in polynomial time. This breaks the security of asymmetric cryptosystems relying on such a problem, like the well-known Rivest-Shamir-Adleman (RSA) algorithm. Similarly, security of Elliptic Curve Cryptography (ECC) relies on the difficulty of discovering the discrete logarithm of a random elliptic curve, which is another problem easily solvable by means of quantum computers.

The above threats has encouraged the cryptographic community to begin migrating to quantum-resistant asymmetric algorithms: this research area is known as Post-Quantum Cryptography (PQC). As evidence of this, in December 2016 the National Institute of Standards and Technology (NIST) started a selection process to define new standards for post-quantum PKC [6]. Initially, 69 candidate cryptosystems were considered, based on various approaches

The associate editor coordinating the review of this manuscript and approving it for publication was Gautam Srivastava.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

arising from different families of hard mathematical problems. One of these problems is that of decoding an arbitrary linear code, which in [7] has been proven to be NP-hard. As arguably the most remarkable example of cryptosystems based on such a problem, we can mention Classic McEliece, which is among the schemes admitted to the third round of the NIST competition [8], and is widely recognized as one of the most secure and promising solutions. Classic McEliece is substantially based on the work of McEliece in 1978 [9], who introduced the first ever proposed code-based cryptosystem employing binary Goppa codes as secret keys. Despite its largely recognized security, the original McEliece scheme relying on Goppa codes has a major drawback in its very large public keys (hundreds of kilobytes for 128 bits of security).

For the above reason, in recent years researchers have put significant efforts to the goal of reducing the public key size of McEliece-like cryptosystems. One of the most promising solutions in this respect consists in replacing the underlying secret code employed in the original McEliece system with a Quasi-Cyclic (QC) random or pseudorandom code [10]. Basically, the idea is that of using codes that admit a compact geometric representation, which leads to a clearly less memory demanding implementation, with respect to unstructured large linear codes (such as Goppa codes). Indeed, such an approach has been used in different NIST submissions, like LEDAcrypt [11], BIKE [12] and HQC [13].

In this work we focus on the implementation of the LEDAcrypt cryptosystem, which uses a class of state-of-the-art error correcting codes named Low-Density Parity-Check (LDPC) codes [14], [15] as secret codes. In its general formulation, the secret LDPC code used in LEDAcrypt is defined by a sparse QC parity-check matrix that can be written as the product of other two QC sparse matrices. Such a structure enables the use of a very efficient decoding algorithm, named *Q-decoder*, which derives from the classical Bit Flipping (BF) decoder [14]. The Q-decoder is specifically tailored for the code structure of LEDAcrypt, and it is significantly faster than classical BF decoding on general LDPC codes (we refer the interested reader to [16, Lemma 2] for more details).

Notice that, while LEDAcrypt was successfully admitted to the first two selection rounds of the NIST PQC competition, it was not admitted to the third round due to the recent discovery of some families of weak keys [17]. However, such an attack is not destructive, and may still be countered with some cautious choices of the system parameters. In this paper, we refer to the LEDAcrypt parameters that were adopted for the second round of the NIST PQC competition. However, we remark that our implementation is completely scalable, i.e., independent of the particularly considered parameters, and thus remains valid also for other versions of LEDAcrypt, that is, characterized by different parameters. Furthermore, we observe that some of the elements that are used in LEDAcrypt (like the ones for multiplications over quotient polynomial rings) are also used in other cryptosystems (such as the aforementioned BIKE and HQC); hence, our results can additionally be employed for other cryptosystems based on pseudorandom QC codes.

A fundamental component of cryptographic functions based on error correcting codes is the decoder. Thus, in this paper we focus on the implementation of the Q-decoder, that is, one of the LEDAcrypt peculiarities, as mentioned above. Actually, the interest for such a decoder goes beyond its cryptographic application, since efficient decoders are also required in conventional coded transmissions, where the main goal is to ensure reliable reception at a reasonable cost (in terms of transmitted power or, equivalently, signal-to-noise ratio). It should be noted, however, that in cryptographic applications impressively low error rates (in the order of 2^{-128} or less) are required to avoid some types of attacks. These values can be reached through a proper design of the LDPC code, able to avoid the appearance of error floor phenomena [16], [18].

A. OUR CONTRIBUTION

Implementations of the Q-decoder have already appeared in the literature [19]–[22]. Moreover, a multiplier specifically designed to handle the arithmetic required in LEDAcrypt has been studied in [23], but it has an area comparable to complete implementations of the LEDAcrypt decoder, like those discussed in [19], [22]. The present work aims to provide an architecture suitable for low-cost implementation on both programmable logic devices, namely Field Programmable Gate Arrays (FPGAs), as well as Application Specific Integrated Circuits (ASICs). The implementation we present in this work extends the scalability and improves the performance of the preliminary implementation in [22]: in particular, in this work we achieve a linear, rather than exponential, growth of the implementation complexity by exploiting parallelism. This study is compared with the most recent prototyping of LEDAcrypt [19], [21], in which the decoder has been implemented trying to reduce the resource utilization or the execution time of the decoding process. We propose an architecture that can achieve a further reduced execution time, while keeping a limited resource utilization. For the proposed ASIC design, we fully characterize our implementation in terms of post-Place&Route area, delay and power figures.

The paper is organized as follows: Section II describes the LEDAcrypt encryption and decryption stages, Section III summarizes the approaches known in the literature to implement a multiplier for cyclic variables, in Section IV the complete architecture of the decoder is presented, in Section V the execution time and area/resource occupation are compared to previous works and, finally, in Section VII some conclusions are drawn.

II. LEDAcrypt

Given an $r \times n$ binary matrix $\mathbf{H}$ called *parity-check matrix*, an $r \times 1$ binary vector $\mathbf{s}$ called *syndrome* and an integer t, the so-called Syndrome Decoding Problem (SDP) consists in expressing $\mathbf{s}$ as the linear combination of no more than t

columns of $\mathbf{H}$. In other words, the SDP consists in finding a $1 \times n$ binary vector $\mathbf{e}$ of Hamming weight not greater than t such that $\mathbf{He}^T = \mathbf{s}$, where ^{T} denotes transposition. For the binary case we consider, in which all variables lie in the binary finite field $\mathbb{F}_2$, the decisional version of the SDP has been proven to be NP-complete in [7]. The $1 \times n$ binary vectors $\mathbf{c}$ corresponding to a null syndrome, i.e., such that $\mathbf{Hc}^T = \mathbf{0}$, are called codewords belonging to the code described by $\mathbf{H}$. Such a code hence corresponds to a linear subspace of $\mathbb{F}_2^n$, where n is the codeword length. Notice that the same code admits multiple representations: in fact, if $\mathbf{H}$ is a valid parity-check matrix for one code, then each $r \times n$ binary matrix $\mathbf{M} = \mathbf{SH}$, where $\mathbf{S}$ is a non-singular $r \times r$ binary matrix, is another valid parity-check matrix for the same code.

Based on the above considerations, each time a codeword is transmitted and no errors occur during transmission, the same codeword is received and the associated syndrome is null. If instead transmission errors occur, they can be modeled as an error vector added to the transmitted codeword, that is, $\hat{\mathbf{c}} = \mathbf{c} + \mathbf{e}$, where $\hat{\mathbf{c}}$ is the vector received upon transmission of $\mathbf{c}$, and the sum is binary. In such a case, we have $\mathbf{H}\hat{\mathbf{c}}^T = \mathbf{H}(\mathbf{c} + \mathbf{e})^T = \mathbf{He}^T = \mathbf{s}$, hence solving the SDP for $\mathbf{s}$ means discovering the error vector $\mathbf{e}$. Then, the transmitted codeword can be recovered as $\mathbf{c} = \hat{\mathbf{c}} + \mathbf{e}$. For this reason, codes of this type are called *error correcting codes*. The solution of the SDP for a given code may be unique up to a certain Hamming weight t. Such a value of t is called the *error correction capability* of the code. For given parameters r and n, the larger the error correction capability, the better the code. While the SDP is exponentially hard for random instances (i.e., for randomly picked parity-check matrices), there exist families of codes for which it can be efficiently solved. Algorithms that aim at solving the SDP are called *decoders*. Code-based encryption schemes, initiated by the seminal works of McEliece [9] and Niederreiter [24], are constructed upon codes of this kind.

In this paper we focus on the Niederreiter framework which, in a nutshell, is represented in Fig. 1 (the asterisk denotes matrix/vector multiplication).

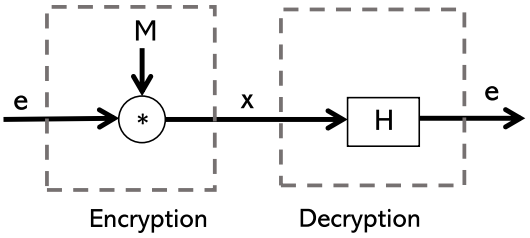


FIGURE 1. Representation of the Niederreiter operating principle.

With reference to the notation introduced in Section I, the SK is the parity-check matrix $\mathbf{H}$ of some error correcting code, which must be equipped with an efficient decoding algorithm, while the PK is derived by applying a secret linear transformation to the secret key. The public key $\mathbf{M}$ obtained through such a transformation must be indistinguishable from the parity-check matrix of a random code. The plaintext message is mapped onto a small weight vector $\mathbf{e}$, which is encrypted into a syndrome via multiplication through the public matrix $\mathbf{M}$. To decrypt, the legitimate user inverts the secret linear transformation used to produce the public key, and uses the decoding algorithm to retrieve the plaintext.

The security of the scheme is based on the fact that an adversary, that does not know the secret code, has no efficient way of decoding (i.e., of finding the plaintext); in an analogous way, the adversary must not be able to retrieve the secret key from the public key; otherwise, he could employ the associated decoder to decrypt the ciphertext. Many families of error correcting codes have failed in fulfilling this last requirement. This is not the case of binary Goppa codes for which, since the first proposal in 1978 [9], no efficient attack is known. Yet, these codes are characterized by remarkably large public keys: for instance, the Classic McEliece NIST PQC candidate, which is based on the Niederreiter framework with binary Goppa codes, requires public keys more than 32 kB long for a 128-bit security level.[1] Given these large public keys, as mentioned in Section I, researchers have focused their effort along the years in trying to find other families of codes, with the goal of reducing the public key size.

One of the most promising solutions in this sense is based on Quasi-Cylic Low-Density Parity-Check (QC-LDPC) codes [25] and Quasi-Cylic Moderate Density Parity-Check (QC-MDPC) codes [26], which require less memory to be stored owing to the QC structure of their characteristic matrices. As also mentioned in Section I, QC-LDPC codes are at the base of the LEDAcrypt cryptosystem. Quasi-cyclicity implies that the code can be described by a parity-check matrix made of circulant blocks, i.e., square matrices in which each row is obtained by cyclically shifting the previous row by one position, as shown in (1) for a generic $p \times p$ circulant matrix $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{p-1} \\ a_{p-1} & a_0 & a_1 & \dots & a_{p-2} \\ a_{p-2} & a_{p-1} & a_0 & \dots & a_{p-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \dots & a_0 \end{bmatrix}. \tag{1}$$

To store a circulant matrix, it is evident that only a row (say, the first one) is required; furthermore, this geometric structure allows for a very efficient algebra, as we will describe in the following sections.

Moreover, in LDPC codes the parity-check matrix is very sparse, i.e., most of its elements are null; this property allows the design of very efficient decoders, with complexity that grows linearly with the code length. The implications of this property will be also discussed below.

[1] We say that a system reaches a λ-bits security level if every algorithm that recovers the secret key or deciphers intercepted ciphertexts has a computational complexity that is not lower than 2^λ.

A. LEDAkem

LEDAcrypt [11] is a suite of cryptographic public key algorithms; it contains LEDApkc, which is a Public-Key Encryption (PKE) algorithm built upon the McEliece framework, and LEDAkem, a Key-Encapsulation Mechanism (KEM) built upon the Niederreiter framework. In this paper we focus on the implementation of LEDAkem with ephemeral keys, i.e., with key-pairs that are refreshed after a single use [11], [27]. The peculiar aspect of the LEDAcrypt version here considered is that, instead of using a single parity-check matrix $\mathbf{H}$, the secret key is composed by two matrices $\mathbf{H}$ and $\mathbf{Q}$, with the following structure

$$\mathbf{H} = [\mathbf{H}_0, \mathbf{H}_1, \ldots, \mathbf{H}_{n_0-1}], \tag{2}$$

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{0,0} & \mathbf{Q}_{0,1} & \cdots & \mathbf{Q}_{0,n_0-1} \\ \mathbf{Q}_{1,0} & \mathbf{Q}_{1,1} & \cdots & \mathbf{Q}_{1,n_0-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{Q}_{n_0-1,0} & \mathbf{Q}_{n_0-1,1} & \cdots & \mathbf{Q}_{n_0-1,n_0-1} \end{bmatrix}. \tag{3}$$

Each $\mathbf{H}_i$ and $\mathbf{Q}_{i,j}$ is circulant; the weight of the columns in $\mathbf{H}_i$ is d_H, while the weights of the blocks in $\mathbf{Q}$ are defined by the matrix

$$\mathbf{W}_Q = \begin{bmatrix} w_0 & w_1 & \ldots & w_{n_0-1} \\ w_{n_0-1} & w_0 & \ldots & w_{n_0-2} \\ \vdots & \vdots & \ddots & \vdots \\ w_1 & w_{n_0-1} & \ldots & w_0 \end{bmatrix}, \tag{4}$$

such that $\mathbf{Q}_{i,j}$ has weight equal to the element in the i-th row and j-th column of $\mathbf{W}_Q$, which is again a circulant matrix with first row $\mathbf{w}_Q = [w_0, w_1, \ldots, w_{n_0-1}]$. We denote with $m = \sum_{i=0}^{n_0-1} w_i$ the sum of the elements in each row and column of $\mathbf{W}_Q$.

To derive the public key, one first computes $\mathbf{L} = \mathbf{HQ} = [\mathbf{L}_0, \mathbf{L}_1, \ldots, \mathbf{L}_{n_0-1}]$, where each block $\mathbf{L}_i$ is again a circulant matrix with size p, such that $n = pn_0$. Once $\mathbf{L}$ has been computed, the weights of its blocks are checked: if cancellations occur, i.e., if the weight of any circulant block in $\mathbf{L}$ is lower than md_H, the secret key is discarded and a new pair of matrices $\mathbf{H}$ and $\mathbf{Q}$ is generated. The matrix $\mathbf{L}$ is then obfuscated as

$$\mathbf{M} = \mathbf{L}_{n_0-1}^{-1}\mathbf{L} = [\mathbf{M}_l, \mathbf{I}_p],$$

where $\mathbf{I}_p$ is the identity matrix of size p. The plaintext is represented by a vector $\mathbf{e}$ with weight t; to encrypt, one simply performs syndrome computation, that is $\mathbf{x} = \mathbf{Me}^T$. To decrypt, one first computes

$$\mathbf{s} = \mathbf{L}_{n_0-1}\mathbf{x} = \mathbf{Le}^T = \mathbf{HQe}^T = \mathbf{H}(\mathbf{eQ}^T)^T,$$

and then exploit the sparsity of $\mathbf{L}$ to retrieve $\mathbf{e}$. The decoder employed in LEDAcrypt is detailed in the next section.

The LEDAkem parameters we have considered in this work are reported in Table 1 [16, Table 2].

TABLE 1. Considered LEDAkem parameters.

Security (in bits)	n_0	p	d_H	$\mathbf{w}_Q$	t
128	2	14,939	11	[4, 3]	136
	3	8,269	9	[4, 3, 2]	86
	4	7,547	13	[2, 2, 2, 1]	69
192	2	25,693	13	[5, 3]	199
	3	16,067	11	[4, 4, 3]	127
	4	14,341	15	[3, 2, 2, 2]	101
256	2	36,877	11	[7, 6]	267
	3	27,437	15	[4, 4, 3]	169
	4	22,691	13	[4, 3, 3, 3]	134

B. THE Q-DECODER

In LEDAkem, the decryption phase finds the low-weight error vector that, in the encryption phase, has been mapped into the syndrome ciphertext; this is realized with an improved BF algorithm [14]. Crucial quantities in a generic BF algorithm are the Unsatisfied Parity Check (UPC) counters, which correspond to the number of unsatisfied parity equations for each element of the unknown error vector. The UPC counters are then employed to locate the errors positions, through a simple threshold criterion: the positions with a counter exceeding the chosen threshold are deemed as error affected. The threshold values selection can be performed in many different ways, depending on the particular BF variant employed.

The BF variant used in LEDAkem is called *Q-Decoder*; it takes into account the particular code structure (i.e., multiplication between $\mathbf{H}$ and $\mathbf{Q}$) to speed-up the decryption phase. The Q-Decoder procedure is detailed in Algorithm 1. The decoder takes as input the syndrome $\mathbf{s}$, that is, the product between the secret circulant block $\mathbf{L}_{n_0-1}$ and the ciphertext $\mathbf{x}$, and either returns an estimate of the error vector $\mathbf{e}$ or reports decoding failure. Decoding is performed through an iterative procedure, that is, the error vector estimate and the syndrome are continuously updated until a null syndrome is obtained or the maximum number of iterations It_{max} is reached. In the first case, decoding has been successful and the algorithm returns the estimated error vector $\mathbf{e}$, otherwise decoding has failed and the algorithm reports the failure event.

Note that, together with $\mathbf{e}$, the decoder updates another vector $\tilde{\mathbf{e}}$, such that it always corresponds to $\mathbf{eQ}^T$. To do this, each time a bit is flipped in $\mathbf{e}$, $\tilde{\mathbf{e}}$ is coherently updated by adding the corresponding row of $\mathbf{Q}^T$ (in the algorithm, $\mathbf{q}_i$ denotes the i-th row of $\mathbf{Q}^T$). This vector is used to update the syndrome at the end of each iteration, by summing the product $\mathbf{H}\tilde{\mathbf{e}}^T$ to the initial syndrome (which is stored as $\mathbf{s}^{(0)}$).

The decisions on the bits to be flipped are taken on the base of the correlation values, collected in a vector $\boldsymbol{\rho}$, which are computed in two steps. The current syndrome $\mathbf{s}$ is first multiplied by $\mathbf{H}$, obtaining $\boldsymbol{\sigma}$. Note that this multiplication is performed in the integer domain, and as such the resulting vector $\boldsymbol{\sigma}$ takes values in $[0; d_H]$. Then, $\boldsymbol{\sigma}$ gets multiplied by $\mathbf{Q}$ to obtain $\boldsymbol{\rho}$, which takes values in $[0; md_H]$ and whose entries provide the UPC counters for the BF algorithm.

Algorithm 1 Q-Decoder

Input: syndrome $\mathbf{s} \in \mathbb{F}_2^p$, parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{p \times n_0 p}$, sparse matrix $\mathbf{Q} \in \mathbb{F}_2^{n_0 p \times n_0 p}$, maximum number of iterations $It_{max} \in \mathbb{N}$

Output: estimated error $\mathbf{e}$ or failure report

$It = 0$ ▷ Iterations counter
$\mathbf{s}^{(0)} = \mathbf{s}$ ▷ Store the initial syndrome in $\mathbf{s}^{(0)}$
$\mathbf{e} = \mathbf{0}, \tilde{\mathbf{e}} = \mathbf{0}$ ▷ Null vectors with length $n_0 p$
while $It < It_{max}$ **or** $\mathbf{s} = \mathbf{0}$ **do**
  $\boldsymbol{\sigma} = \mathbf{s}^T \star \mathbf{H}$
  $\boldsymbol{\rho} = [\rho_0, \ldots, \rho_{n-1}] = \boldsymbol{\sigma} \star \mathbf{Q}$
  $b = f(\mathbf{s})$ ▷ Compute threshold
  $P = \{i \in [0, n-1] | \rho_i > b\}$
  for $i \in P$ **do**
    $e_i = e_i \oplus 1$ ▷ Update error vector estimate
    $\tilde{\mathbf{e}} = \tilde{\mathbf{e}} \oplus \mathbf{q}_i$
  end for
  $\mathbf{s} = \mathbf{s}^{(0)} \oplus \mathbf{H}\tilde{\mathbf{e}}^T$ ▷ Update syndrome
  $It = It + 1$ ▷ Increase iterations counter
end while
if $\mathbf{s} = \mathbf{0}$ **then**
  return $\mathbf{e}$ ▷ Decoding is successful
else
  Report failure ▷ Decoding has failed
end if

It is important to note that the encrypted message, the syndrome and all the matrices are binary while, as mentioned, $\boldsymbol{\sigma}$ and $\boldsymbol{\rho}$ are vectors of integers. To evidence multiplications that are computed in the integer domain, in Algorithm 1 we have used the operator $\star$. In analogous way, the operator $\oplus$ is employed to indicate sums that are performed in the binary finite field.

In LEDAkem, the threshold, noted by b, is chosen according to a law $f(\mathbf{s})$, that is a piece-wise function of the syndrome weight. Such a function, which depends on the code size and on the desired correction capabilities of the decoder, is efficiently stored as a Look-Up Table (LUT) filled with pairs (w_i, b_i), where $w_i \in [0; p]$ represents the syndrome weight and $b_i \in [\lceil md_H/2 \rceil ; md_H]$ is the associated threshold value. On input $\mathbf{s}$, the function finds the largest w_i, among those that are lower than the weight of $\mathbf{s}$, and returns the associated threshold value b_i. For more details about how the LUT is built, we refer the reader to [27, Section 2.4]. On the base of the chosen threshold, the error estimate and the syndrome are coherently updated.

It can be easily seen that ρ_j (i.e., the j-th entry of vector $\boldsymbol{\rho}$) is obtained by summing the entries of $\mathbf{s}$ that are indexed by set entries in the j-th column of $\mathbf{L}$. Hence, the values in $\boldsymbol{\rho}$ correspond to the UPC counters. However, as we will discuss afterwards, the Q-decoder exploits the particular geometry of the secret $\mathbf{L}$ and performs the counters computation through a two-steps procedure; by doing this, the execution time gets significantly reduced, with respect to a traditional BF decoder.

C. COMPUTATIONAL COMPLEXITY AND COMPARISON WITH OTHER DECODERS

In principle, any LDPC or MDPC decoder can be used in the decryption phase of LEDAkem.[2] Indeed, all such decoding techniques are solely based of the sparseness of the employed parity-check matrix, a characteristic that holds true also for the codes employed in LEDAkem. Recent works have proposed and analyzed decoders for generic MDPC codes (for instance, see [28]–[31]). All of such algorithms employ a very low latency, iterative procedure, where in each iteration some common operations (i.e., counters computation and syndrome update) are performed: the difference between different techniques lies in how the counters are processed (i.e., how error affected positions are located).

As we have already said, any MDPC decoder may replace the Q-decoder in the LEDAkem decryption phase. However, these algorithms are designed to be used for generic MDPC codes, and hence do not take into account the particular product structure of the secret parity-check matrix in LEDAkem. This is where the peculiarity of the Q-decoder comes into play: it integrates the factorization of the parity-check matrix $\mathbf{L}$ into $\mathbf{H}$ and $\mathbf{Q}$, to speed up the decoding procedure. By doing this, one can easily prove that the counters computation and each syndrome update come with a cost of $O\big((m + d_H)n\big)$ and $O\big(m + d_H\big)$ elementary operations, respectively [16, Lemma 2].

For an MDPC code with column weight v, the counters computation is done with $O(nv)$ elementary operations, while for each flipped bit the syndrome update is performed with a cost of $O(v)$. In LEDAkem, the secret key $\mathbf{L}$ has columns of weight $v = md_H$; hence, we can roughly estimate the computational advantage of the Q-decoder (with respect to generic MDPC decoders) as

$$\frac{md_H + 1}{m + d_H + 1} \approx \left(m^{-1} + d_H^{-1}\right)^{-1}.$$

To make a numerical example, let us consider the LEDAkem parameters for the 128-bits security instance with $n_0 = 2$: since $d_H = 11$ and $m = 7$, we have $m^{-1} + d_H^{-1} = 0.234$, implying that the Q-decoder is expected to run approximately 4 times faster than MDPC decoding schemes.

Taking the above cost estimates into account, as in [16], the computational complexity of the Q-decoder can be assessed as

$$O\Big(nIt_{max}(m + d_H) + (m + d_H + 1)t\Big).$$

Notice that the above cost has been derived under the assumption that the Q-decoder performs exactly t bit flips, i.e., that no bit is wrongly estimated as error affected. In practice, one observes that the decoder always makes a very limited

[2]The term MDPC, introduced in [26], refers to a code which can be described by a parity-check matrix whose density is only slightly higher than that of a typical LDPC code. Hence, the matrix is still somehow sparse. From a decoding perspective, there is no meaningful difference between the two families of codes and, hence, they can be decoded through the same techniques.

number of wrong flips, so that the above estimate is always quite accurate.

III. CIRCULANT MATRIX PRODUCT

The multiplication of a vector by a circulant matrix is a recurrent operation in LEDAcrypt: many processing tasks (such as encrypting, computing **s** and the correlation values) are indeed the result of a circulant matrix product.

To describe this operation, we can consider the product $\mathbf{r} = \mathbf{vA}$, with $\mathbf{v} = [v_0, v_1, \ldots, v_{p-1}]$ being a length-p row vector and $\mathbf{A}$ a square circulant matrix of size p (i.e., with structure as in (1)). The result is the length-p vector $\mathbf{r} = [r_0, r_1, \cdots, r_{p-1}]$ where each entry r_i is obtained as

$$r_i = \sum_{j=0}^{p-1} v_j a_{\,\mathrm{mod}\,(i-j,p)}.$$

Notice that the result is the same as the carry-less product between integers or the polynomial multiplication evaluated in $\mathbb{F}_2[x]/(x^p+1)$ (see [27] for more details about the relation between operations with circulant matrices and the algebra of polynomials).

The most common method to evaluate **r** is the Schoolbook algorithm. However, this should be adapted in order to efficiently implement the multiplications present in LEDAcrypt. In particular, we must note that the multipliers for LEDAcrypt involve vectors with size of several thousands (see the parameter p in Table 1). Hence, it becomes important to further explore implementations of multipliers that can compute circulant matrix products in an efficient way.

A. STATE OF THE ART

A multiplier for large integer or polynomial multiplications can be efficiently implemented by means of two main approaches: the Karatsuba and Ofman [32] and the Schönhage–Strassen [33] algorithms. To compare the algorithms, we employ the *time complexity metric*, that is, the number of required binary operations as a function of the input length p. As shown in Fig. 2, the complexity of the Schoolbook algorithm grows as p^2, while for the Karatsuba and Schönhage–Strassen multiplications it evolves as $p^{\log_2(3)}$ and $p\log_2(p)\log_2\left(\log_2(p)\right)$, respectively [33].

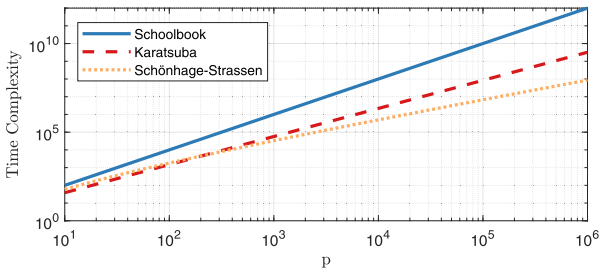


FIGURE 2. Comparison between the time complexity of multiplication algorithms, as a function of the input size *p*.

While Karatsuba and Schönhage–Strassen are generally faster than the Schoolbook multiplication, they are however characterized by a larger hardware complexity [23], [34], [35]. Indeed, while the Schoolbook algorithm uses only a large adder, the Karatsuba multiplier (in its basic version) employs two adders and a small multiplier, while Schönhage–Strassen requires a multiplier plus a Fast Fourier Transform (FFT) module. Hence, as discussed in [36], more logic elements are necessary to implement Karatsuba and Schönhage–Strassen multipliers than the Schoolbook multiplier.

In the present study we use a simplified version of the Schoolbook multiplier, and we adapt it to the case of very sparse circulant matrices. We will refer to the resulting multiplier as Vector by Sparse Circulant (VbSC).

In particular, VbSC reduces the time complexity from p^2 to pd, where d is the number of ones in the first row (or column) of the matrix. The sparsity of the matrix can reduce the Schoolbook multiplier time complexity, up to the point that, for very sparse matrices (as those employed in LEDAcrypt) its running time becomes comparable to that of the Karatsuba and Schönhage–Strassen algorithms. Notice that a similar idea has already been exploited in QcBits [37], for software implementation of a method to multiply moderately dense matrices.

In this work, we adapt such modified multiplication method for hardware implementation, with the purpose of further reducing both the execution time and the area occupation.

B. VECTOR BY SPARSE CIRCULANT

The way the Schoolbook algorithm evaluates $\mathbf{r} = \mathbf{vA}$ is similar to the canonical vector by matrix multiplication. However, due to the sparsity of the employed matrix (i.e., most of its elements are null), we can avoid to compute many elements, since they will be null: this way, we can save a significant amount of time.

Moreover, the execution can be further simplified since the presence of a circulant matrix reduces the result to the sum of cyclic rotations of the input vector **v**. These rotated input vectors are referred to as partial products, $\mathbf{v}^{(i)}$, with $i = 0, 1, \ldots d_A - 1$, where d_A corresponds to the number of non zero elements in the first row (or any other row) of $\mathbf{A}$.

To clarify the derivation of our proposed algorithm, we show an example for the case of $p = 5$. The components of the vector r are calculated as

$$\begin{aligned}
r_0 &= v_0a_0 + v_1a_1 + v_2a_2 + v_3a_3 + v_4a_4,\\
r_1 &= v_0a_4 + v_1a_0 + v_2a_1 + v_3a_2 + v_4a_3,\\
r_2 &= v_0a_3 + v_1a_4 + v_2a_0 + v_3a_1 + v_4a_2,\\
r_3 &= v_0a_2 + v_1a_3 + v_2a_4 + v_3a_0 + v_4a_1,\\
r_4 &= v_0a_1 + v_1a_2 + v_2a_3 + v_3a_4 + v_4a_0. \qquad (5)
\end{aligned}$$

If sparsity is included, for example with $a_0 = a_2 = a_3 = 1$ and $a_1 = a_4 = 0$, after a rearrangement of the output,

the result becomes

$$\begin{aligned} r_0 &= v_0 + v_2 + v_3 = v_0 + v_2 + v_3, \\ r_1 &= v_1 + v_3 + v_4 = v_1 + v_3 + v_4, \\ r_2 &= v_0 + v_2 + v_4 = v_2 + v_4 + v_0, \\ r_3 &= v_0 + v_1 + v_3 = v_3 + v_0 + v_1, \\ r_4 &= v_1 + v_2 + v_4 = v_4 + v_1 + v_2, \end{aligned} \tag{6}$$

The example clearly shows that it is enough to generate the rows by rotating the input. This remark is helpful for large p.

The formal description of the VbSC is shown in Algorithm 2. The input $\mathbf{v}$ is stored as a $1 \times p$ vector, while to represent the matrix $\mathbf{A}$ we use the set S_A containing the positions of asserted entries in the first row. The quantities d_A and $S_A(i)$ denote the length of the position vector and the i-element of S_A, respectively.

Algorithm 2 Vector by Sparse Circulant (VbSC)

Input: length-p vector $\mathbf{v}$, weight of $\mathbf{A}$ (interpreted as the weight of each row and column) $d_A \in \mathbb{N}$, first row of $\mathbf{A}$ represented as a list of positions S_A with size d_A

Output: length-p vector $\mathbf{r} = \mathbf{vA}$

1: $\mathbf{r} = \mathbf{0}$ ▷ Initialized as null vector with length p
2: **for** $i = 0$ **to** $d_A - 1$ **do**
3: $k = S_A(i)$
4: $\mathbf{v}^{(i)} = [v_k, v_{k+1}, \cdots, v_{p-1}, v_0, v_1, \cdots, v_{k-1}]$
5: $\mathbf{r} = \mathbf{r} + \mathbf{v}^{(i)}$ ▷ Update $\mathbf{r}$ with new partial product
6: **end for**
7: **return r**

Clearly, the advantage of this approach is in the generation of the partial products: one iteration of the algorithm is enough to obtain one partial product. Notice that, when $\mathbf{A}$ is a matrix containing only zeros and ones, with the same algorithmic structure we can perform both the integer multiplication (i.e., $\mathbf{v} \star \mathbf{A}$) and the canonical product over the binary finite field (that is, $\mathbf{vA}$). Indeed, when both $\mathbf{v}$ and $\mathbf{A}$ are defined over the finite field $\mathbb{F}_2$, it is enough to update $\mathbf{r}$ by xor-ing it with each partial product $\mathbf{v}^{(i)}$.

The hardware implementation of the multiplier is described in the following sections.

1) MEMORY

The efficient mapping of memories onto physical components available in FPGA devices or ASICs is a critical aspect, because of the large values of p in Table 1.

Let n_b, a power of two, denote the machine word size (i.e., the register size for the chosen architecture). The input and output vectors of VbSC are divided into h words, with $h = \lceil p/n_b \rceil$, as follows

$$\mathbf{r} = [\underbrace{r_0 \cdots r_{n_b-1}}_{\substack{\#\,0 \\ n_b \text{ entries}}} \cdots \underbrace{r_{(h-2)n_b} \cdots r_{(h-1)n_b-1}}_{\substack{\#\,(h-1) \\ n_b \text{ entries}}} \underbrace{r_{(h-1)n_b} \cdots r_{p-1}}_{\substack{\#\,h \\ p-(h-1)n_b \text{ entries}}}].$$

The first $h-1$ words contain n_b entries, while we use the last one to store the remaining $p-(h-1)n_b$ entries. Since p must be a prime, in order to avoid cryptanalysis exploiting factorization [27], the last word will always represent a number of entries that is lower than n_b, and the remaining elements of the register will be filled with zeros.

Hence, in the design we have that the input and the result vectors, $\mathbf{v}$ and $\mathbf{r}$, are stored as matrices of size $h \times n_b$, which we refer to as M_v and M_r, respectively. The element in position (i, j) in the matrix corresponds to the vector entry in position $in_b + j$.

To store the matrix $\mathbf{A}$, as already mentioned, we exploit the fact that it is sparse and hence we represent it as a list of positions.

In particular, we can consider only the first row of $\mathbf{A}$, and just represent its d_A asserted positions. To do this, we employ a format that is analogous to that for $\mathbf{r}$ and $\mathbf{v}$, and hence use two memories M_{adx} and M_{shift}. For each position, $\lceil \log_2(p) \rceil$ bits are required: the first $\lceil \log_2(p) \rceil - \log_2(n_b)$ are stored in M_{adx}, while for the last $\log_2(n_b)$ bits we use M_{shift}. Finally, we use $M_{shift}(i)$ and $M_{adx}(i)$ to denote the access to the representation of the i-th asserted entry in the first row of $\mathbf{A}$ (so, we have $0 \le i \le d_A - 1$).

2) EXECUTION

The architecture of VbSC generates n_b elements of each partial product $\mathbf{v}^{(i)}$ (for $0 \le i \le d_A - 1$) in a single iteration. The starting point is the i-th asserted element in the first row of $\mathbf{A}$, whose position is stored through $M_{adx}(i)$ and $M_{shift}(i)$. Remember that $\mathbf{v}$ is represented as an $h \times n_b$ matrix M_v, and that $\mathbf{v}^{(i)}$ is obtained by rotating the input $\mathbf{v}$ by the value of $S_A(i)$ (see Algorithm 2). The rotation is realized in two steps: we initially rotate the rows of M_v by the amount stored in $M_{adx}(i)$, and then apply a final column-shift of $M_{shift}(i)$. In details, let M_v^i denote the matrix representation for $\mathbf{v}^{(i)}$; its j-th row is obtained from the rows of M_v with indices $\mathrm{mod}(M_{adx}(i) + j, h)$ and $\mathrm{mod}(M_{adx}(i) + j + 1, h)$: the rows are loaded in memory in a $2n_b$ large register and connected to the *collapse unit* together with $M_{shift}(i)$. At this point the unit extracts the portion of bits of the result, which is loaded in memory, too. In the next cycle, $\mathrm{mod}(M_{adx}(i) + j + 2, h)$ row is loaded, $\mathrm{mod}(M_{adx}(i) + j, h)$ discarded and $\mathrm{mod}(M_{adx}(i) + j + 1, h)$ moved in the first position of the $2n_b$ register; then, extraction and load in M_r are completed. The process goes on until the complete $\mathbf{v}^{(i)}$ is generated. For $i = 0$, we simply load the words of the partial product $\mathbf{v}^{(0)}$ in memory; for the remaining values of i, each generated word is xor-ed with the value already present in memory.

Figure 3 provides an example of the VbSC execution, for the case of $h = 6$ and $n_b = 4$. The two consecutive rows are read from memory and loaded in the datapath; the portion of $\mathbf{v}^{(i)}$ to extract is highighted in M_v; the update of the result is done reading the j-th row of M_r and then xor-ed with the current $\mathbf{v}^{(i)}$ row. Finally, the j-th word of M_r is updated. These two operations on the words of M_r are highlighted in Fig. 3 with a lighter and darker blue color, respectively.

The function of the collapse unit is to extract the target n_b bits from the initially loaded 2 n_b bits. In [22], this unit

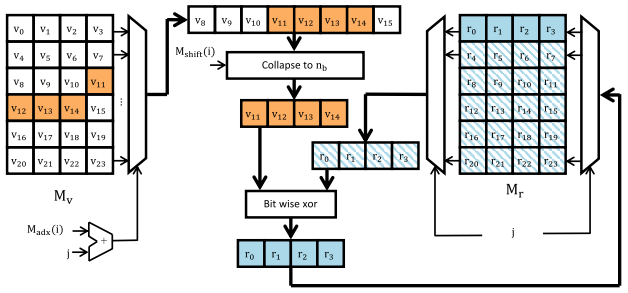


FIGURE 3. Detailed representation of the VbSC execution, for $h = 6$ and $n_b = 4$.

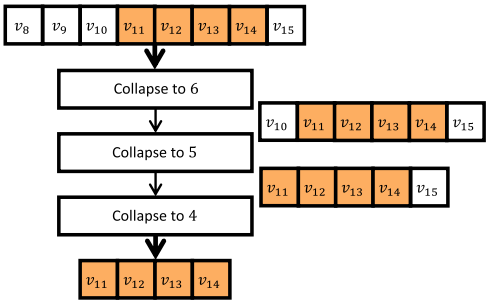


FIGURE 5. Rotation by three positions, for the case $n_b = 4$.

has been implemented as a single multiplexer that takes as input all the possible p bits which can be selected from $\mathbf{v}$. In this work we have considered a different version for the multiplexer, which we call *logarithmic version*: the rotation is organized in $\log_2(n_b)$ levels, such that the ℓ-th level (where ℓ goes from $\log_2(n_b)$ to 0) operates a rotation of $n_b/2^{\ell-1}$, plus one rotation at the end. This strategy results in a cascade of $\log_2(n_b) + 1$ multiplexers with only two inputs. In Fig. 4 the example for one layer of collapse is shown: the choice, made through s, is to reduce the number of bits from $2n_b$ to $n_b+n_b/2$ by taking the first $n_b+n_b/2$ bits or the last $n_b+n_b/2$ bits; these operations correspond either to a rotation by $n_b/2$, or by 0 positions. The rotation that covers a wide range of possibilities is realized with a series of cascaded collapse units. The example in Fig. 5, for $n_b = 4$, shows how one can perform a rotation of at most four positions. This is achieved with three layers (since $\log_2(4) + 1 = 3$); each layer has a two input multiplexer that selects between a rotation of 0 or $n_b/2^l$ and its output is the input of the next multiplexer level.

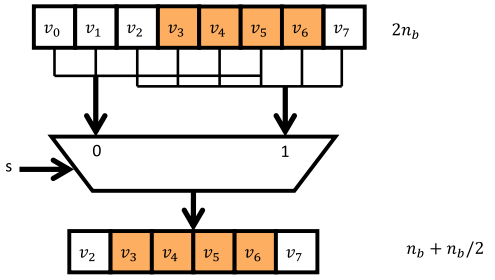


FIGURE 4. The collapse unit reduces $2n_b$ bits to $n_b + n_b/2$ bits.

The total number of clock cycles required to compute a multiplication can be easily derived as

$$N_{cycles}^{VbSC}(h, d_A) = 3hd_A + 2d_A. \tag{7}$$

Each word of the result is computed in three clock cycles, as it requires to load the rows of M_v, extract $\mathbf{v}^{(i)}$ and then to load the updated M_r row (each of these operations is performed in just one clock cycle). Furthermore, we have to take into account the cost of two loads from memory: the rows from M_{adx}, together with M_{shift}, and the first read from M_v.

C. SPARSE VECTOR BY SPARSE CIRCULANT

The unit VbSC can be further improved, in order to reduce the time complexity, when the input vector is also sparse. To show how the multiplication gets simplified in this case, let us refer again to the example in (6), and further assume that v_0 and v_4 are the only non null entries in $\mathbf{v}$. This yields

$$\begin{aligned} r_0 &= v_0 + 0 + 0 = v_0, \\ r_1 &= 0 + 0 + v_4 = v_4, \\ r_2 &= v_0 + 0 + v_4 = v_0 + v_4, \\ r_3 &= v_0 + 0 + 0 = v_0, \\ r_4 &= 0 + 0 + v_4 = v_4. \end{aligned} \tag{8}$$

The result in (8), with respect to that in (6), requires a reduced number of evaluations. More in general, for a vector $\mathbf{v}$ that has d_v non zero elements and a circulant matrix $\mathbf{A}$ with rows and columns with weight d_A, the number of elements to generate for the resulting $\mathbf{r} = \mathbf{vA}$ is only $d_v d_A$. In this case, one can compute products through the approach we report in Algorithm 3.

Algorithm 3

Input: weight of $\mathbf{v}$ $d_v \in \mathbb{N}$, weight of $\mathbf{A}$ (interpreted as the weight of each row and column) $d_A \in \mathbb{N}$, vector $\mathbf{v}$ represented as a list of positions S_v with size d_v, first row of $\mathbf{A}$ represented as a list of positions S_A with size d_A

Output: length-p vector $\mathbf{r} = \mathbf{vA}$

```
r = 0                          ▷ Initialize r as a null vector with length p
for iA = 0 to d_A − 1 do
    for iv = 0 to d_v − 1 do
        ir = mod (S_v(iv) + S_A(iA), p)
        r_ir = r_ir + 1        ▷ Update entry in position ir
    end for
end for
return r
```

Sparse Vector by Sparse Circulant (SVbSC) is an almost direct mapping of Algorithm 3. The only difference is in the evaluation of mod(a, p): it is split in two cycles in order to speed up the execution.

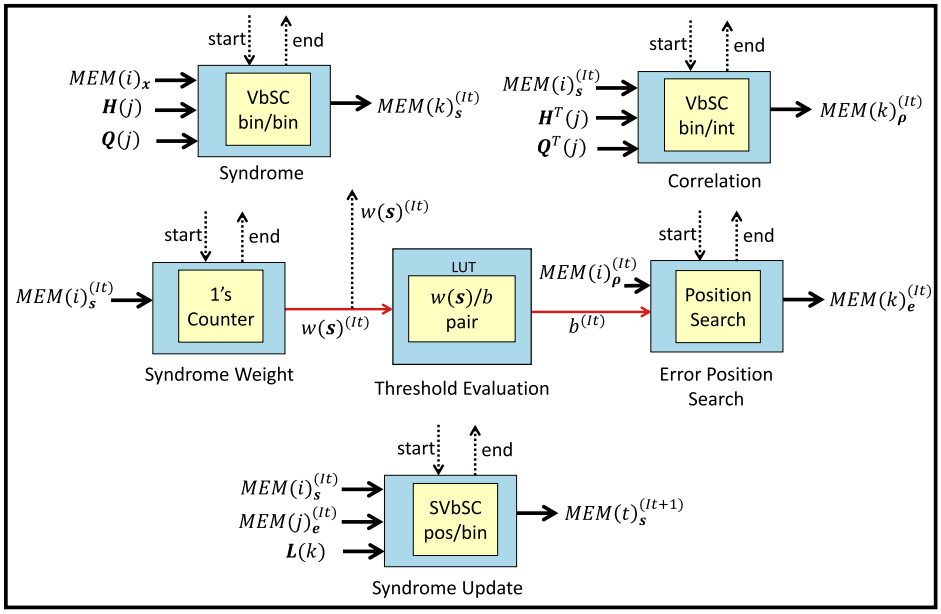


FIGURE 6. Schematic representation of the Data Path, with its elements and connections.

Actually, it can be easily proved that the total number of cycles becomes

$$N_{cycles}^{SVbSC}(d_A, d_v) = d_A + d_v + 6d_A d_v. \tag{9}$$

IV. DECODER ARCHITECTURE

The decoder architecture consists of three main sections, the Data Path, the Control Unit and the Memory block. The multipliers are key components in the Data Path, exploited to compute and update $\mathbf{s}$ and $\boldsymbol{\rho}$ in Algorithm 1. One relevant element of flexibility in the decoder is the design-time selection of the degree of parallelism, which ranges from 8 to 256. The choice reduces the overall execution time thanks to the presence of parallelizable units, but is limited by the components that do not have such feature. As discussed next, this aspect has been analyzed in detail and two versions of the decoder have been derived.

A. DATA PATH

In this section we analyze each internal unit of the Data Path, which is the most complex component of the decoder. The high-level view of this component is depicted in Fig. 6, where the connections with Memory, Control Unit and inner units are highlighted; precisely, bold lines connect the Memory to the inner units, dotted lines connect the Data Path with the Control Unit, while red lines are used for internal signals. The Data Path includes a dedicated unit for each key processing part of the Q-decoder (Algorithm 1), which we divide as Syndrome (computation of $\mathbf{s}$ which is given as input to the decoder), Correlation (computation of the vectors $\boldsymbol{\sigma}$ and $\boldsymbol{\rho}$), Syndrome Update (computation of the syndrome which is given as input to the next iteration), Error Position Search (i.e., determination of the positions in the estimated $\mathbf{e}$ that need to be updated), Syndrome Weight together with Threshold Evaluation (threshold computation through the LUT based on the current syndrome weight). In the remainder of this section, we describe each unit and its execution time, which we express in terms of required number of cycles. Notice that such a quantity depends on the code parameters (i.e., n_0, p and the weights of the blocks in the matrices $\mathbf{H}$ and $\mathbf{Q}$), as well as the word size n_b, which determines the value of $h = \lceil p/n_b \rceil$. Furthermore, some units have a running time that may change depending on some characteristic parameters of the considered iteration (such as the number of performed flips); to highlight this dependence, where present, we will use the iteration counter (It) as a superscript.

1) SYNDROME AND CORRELATION

These units are connected to the input and output memories, which are addressed by the VbSC module. Since the Syndrome and Correlation units work on different types of input and output vectors, each unit employs its own VbSC multiplier.

To evaluate $\mathbf{s}$, consider that

$$\mathbf{s} = \mathbf{L}_{n_0-1}\mathbf{x} = \sum_{i=0}^{n_0-1} \mathbf{H}_i \left(\mathbf{Q}_{i,n_0-1}\mathbf{x}\right),$$

where each $\mathbf{H}_i$ has weight d_H while, recalling (4), $\mathbf{Q}_{i,n_0-1}$ has weight w_{n_0-1-i}. Since $\mathbf{x}$ is dense, we first use VbSC to compute the products $\mathbf{Q}_{i,n_0-1}\mathbf{x}$, and then use it again to multiply each product by $\mathbf{H}_i$. So, we measure the number of

cycles to compute $\mathbf{s}$ as

$$N_{Syn} = n_0 N^{VbSC}_{cycles}(h, d_H) + \sum_{i=0}^{n_0-1} N^{VbSC}_{cycles}(h, w_i). \quad (10)$$

Recalling Algorithm 1, we have that since the product $\mathbf{HQ}$ leads to a matrix $\mathbf{L}$ with full weight (i.e., no cancellations occur), the following relation holds

$$\boldsymbol{\rho} = \mathbf{s}^T \star \mathbf{L} = \left(\mathbf{s}^T \star \mathbf{H}\right) \star \mathbf{Q} = \boldsymbol{\sigma} \star \mathbf{Q}.$$

Such a computation is performed in two steps. First, we compute the integer product $\boldsymbol{\sigma}$ using for n_0 times the VbSC algorithm, considering as circulant matrices the blocks in $\mathbf{H}$ (each with weight d_H). Then, we use VbSC again for n_0^2 times, to compute the integer product $\boldsymbol{\sigma} \star \mathbf{Q}$. Hence, we estimate the number of cycles as

$$N_{Corr} = n_0 \left(N^{VbSC}_{cycles}(h, d_H) + \sum_{i=0}^{n_0-1} N^{VbSC}_{cycles}(h, w_i) \right), \quad (11)$$

where, as in (4), w_i refers to the weights of the blocks in $\mathbf{Q}$.

2) SYNDROME UPDATE

The Syndrome Update unit performs the multiplication between the matrix $\mathbf{L} = \mathbf{HQ}$ and the estimated error vector $\mathbf{e}^T = [\mathbf{e}_0^T, \mathbf{e}_1^T, \ldots, \mathbf{e}_{n_0-1}^T]$.

We remark that, in principle, both VbSC and SVbSC can be used for such operation. Actually, we must remember that the asserted entries of $\mathbf{e}$ are assessed as the iterations go on, and that $\mathbf{e}$ starts as the null vector. Hence, its Hamming weight likely grows as the iterations go on, it is always rather sparse and reaches the value $t \ll n_0 p$ at the end of the decoding procedure. Hence, we may rely on SVbSC to perform the syndrome update. Let us consider the It-th iteration, and suppose that the weight of the i-th block in $\mathbf{e}$ is $w_{e,i}^{(It)}$. Since each block in $\mathbf{L}$ has weight md_H, we have that the required number of cycles for this unit, when SVbSC is used, is given by

$$N^{SVbSC}_{SynUpt}(It) = \sum_{i=0}^{n_0-1} N^{SVbSC}_{cycles}\left(md_H, w_{e,i}^{(It)}\right). \quad (12)$$

Notice that we expect to have $\sum_{i=0}^{n_0-1}(w_{e,i}^{(It)}) \leq t$, so that as a very rough but reliable upper bound on the number of cycles, independently of the iteration number, we can use $n_0 N^{SVbSC}_{cycles}(md_H, t)$.

In principle, we can also choose to neglect the sparsity of $\mathbf{e}$, and perform the syndrome update via the VbSC unit. By doing this, we obtain a number of cycles that is given by

$$N^{VbSC}_{SynUpt} = n_0 N^{VbSC}_{cycles}(h, m + d_H). \quad (13)$$

We notice that, in this case, the cost of updating the syndrome is independent of the iteration number (since it does not depend on the weight of the current $\mathbf{e}$).

3) ERROR POSITION SEARCH

The Error Position Search unit evaluates the error positions from the correlation values and stores them in the address (rows) and shift (column) format, in order to make the positions available to the SVbSC multiplier. This search requires to read the Correlation vector, and to compare its entries with the threshold value: if there are no matches, the next row is processed; in case of a match, the complete row is read to store all error positions (n_b elements are processed, thus $n_b + 1$ cycles are required to read the elements and store the positions).

Hence, if we denote with $w_e^{(It)} = \sum_{i=0}^{n_0-1} w_{e,i}^{(It)}$ the weight of $\mathbf{e}$ in the It-th iteration, we have that the required number of cycles is

$$N_{Err}(It) = 2h + (n_b + 1) w_e^{(It)}. \quad (14)$$

4) SYNDROME WEIGHT AND THRESHOLD EVALUATION

The Syndrome Weight unit reads each line of the Syndrome memory and counts the number of asserted entries. The resulting weight, $w(\mathbf{s})^{(It)}$, is used to address the LUT containing the thresholds, in order to obtain the current threshold $b^{(It)}$.

A number h of cycles is necessary to read the rows from the Syndrome memory, while additional h cycles are used to count the number of asserted entries. Finally, two cycles are needed to load the threshold. Therefore, the resulting overall number of cycles is

$$N_{SynW+Th} = 2h + 2. \quad (15)$$

B. CONTROL UNIT

The Control unit provides the start signals to the units in the Data Path and collects their end signals in order to properly synchronize the operations and the syndrome weight $w(\mathbf{s})^{(It)}$, in order to decide whether to end or not the algorithm. The scheduling of the operations is shown in Fig. 7 for an example with $p = 8,269$, $n_0 = 3$, $n_b = 8$ and three decoding iterations.

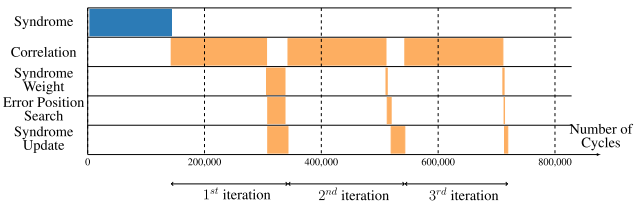


FIGURE 7. Time evolution of the decoding process. On the y-axis the operations performed by the Decoder are listed.

In general, at the end of each iteration, the value of the syndrome weight, $w(\mathbf{s})^{(It)}$, is checked and if it is not equal to zero and the maximum number of iterations has not been reached then the next iteration is started. In the figure, the same (orange) color is used for the successive processing steps in the Q-decoder while a different (blue) color is used for the initial evaluation of the syndrome.

We now derive an estimate on the number of cycles which are necessary to perform a full execution of the Q-decoder. We are going to use It_{dec} to denote the number of performed iterations; notice that we are not able to predict in advance its value, and can thus only claim that, in the worst case, it is equal to It_{max}. Since the execution is sequentially scheduled, the execution time for each iteration is obtained by summing over the times taken by each unit. In particular, we consider two versions for the decoder, which only differ in the unit which is used to perform the syndrome update:

- In version 1 (v1), for all iterations we perform the syndrome update with the SVbSC unit. Hence, for the It-th iteration, we estimate the number of cycles as

$$N_{Iter,I}(It) = N_{Corr} + N_{SynW+Th} + N_{Err}(It) + N^{SVbSC}_{SynUpt}(It). \quad (16)$$

  We notice that the terms N_{Corr} and $N_{SynW+Th}$ only depend on the code parameters, while $N_{Err}(It)$ and $N^{SVbSC}_{SynUpt}(It)$ additionally depend on the iteration number;
- In version 2 (v2), we use both SVbSC and VbSC units to perform the syndrome update at the end of each iteration. When the VbSC unit is employed for the syndrome update in the It-th iteration, we have the following cost for the iteration

$$N_{Iter,II}(It) = N_{Corr} + N_{SynW+Th} + N_{Err}(It) + N^{VbSC}_{SynUpt}. \quad (17)$$

  We notice that, for this version, the only iteration dependent term is $N_{Err}(It)$. Hence, in this decoder version, we switch from one choice to the other, in order to improve the decoder performances (in terms of number of required cycles). To decide between the two possibilities, it is enough to derive the conditions upon which the approach based on VbSC performs better than that based on SVbSC. Using (12) and (13), one easily finds that the SVbSC based version is more convenient if the current vector $\mathbf{e}$ has weight $w_e^{(It)}$ such that

$$w_e^{(It)} < \frac{n_0(m + d_H)(1 + 3h)}{1 + 6n_0\, md_H}.$$

  Starting from this consideration and having realized a series of simulations, we have found that using SVbSC is more convenient if $n_b < 64$, independently of the iteration number, or if $n_b \geq 64$ and $It \geq 2$. In the other cases, relying on VbSC offers better performances.

Given the above considerations, we are ready to derive an estimate for the number of cycles that are performed by a full Q-decoder execution. For version 1, we can estimate the overall number of cycles as

$$N_{dec,I} = N_{Syn} + \sum_{It=0}^{It_{dec}-1} N_{Iter,I}(It), \quad (18)$$

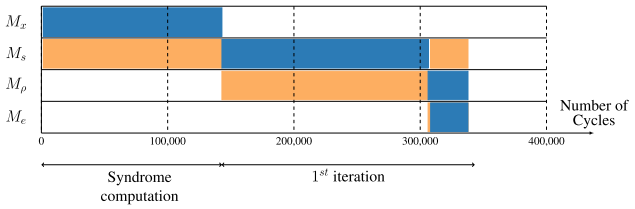


FIGURE 8. The memory accesses during the execution of a decoding step. On the y-axes the memories are listed.

while for version 2 (assuming $n_b \geq 64$), we have

$$N_{dec,II} = N_{Syn} + \sum_{It=0}^{1} N_{Iter,II}(It) + \sum_{It=2}^{It_{dec}-1} N_{Iter,I}(It) \quad (19)$$

where $N_{Iter,I}$ and $N_{Iter,II}$ are given by (16) and (17), respectively. Notice that, in order to derive a closed form theoretical estimate, we would need to know in advance both the number of performed iterations, as well as how the weight of $\mathbf{e}$ evolves throughout the iteration and distributes among the blocks in $\mathbf{e}$. These quantities are hardly to predict and a probabilistic analysis would produce only a rough estimate of the relevant quantities. For this reason, in Section V we will present results obtained through, more significant, experimental evaluations.

C. MEMORY

The Memory unit includes the components employed to store the decoder variables. The dense vectors are stored in the matrix memory format (i.e., divided into words of size n_b), and each block vector is considered as a distinct memory. The matrices are stored as a list of positions. The main memory components are: $M_{\mathbf{x}}$ storing the ciphertext, $M_{\mathbf{s}}$ for the syndrome, M_ρ for the correlation and $M_{\mathbf{e}}$ for the error. Additional memories are allocated to store matrices $\mathbf{L}$, $\mathbf{H}$ and $\mathbf{Q}$ and their transposes.

The accesses to the memory in each phase of the decoding process are shown in Fig. 8, where read (dark areas) and write (light areas) operations are reported for each memory component along one iteration.

V. SYNTHESIS RESULTS

The architecture has been implemented by using Design Compiler® with two different technologies: STM FDSOI 28 nm and UMC CMOS 65 nm. The FPGA implementation has been carried out with Vivado®Design Suite HLx, targeting the Xilinx Artix -7 xc7a200tfbg484-2 device, which is large enough to support the most resource demanding version of the decoder. The results have been obtained for all the possible conditions of parallelism, for given security levels and values of n_0.

A. EXECUTION TIME

The decoder total execution time is computed as

$$T_{decrypt} = \frac{N_{cycles}}{f_{max}}, \quad (20)$$

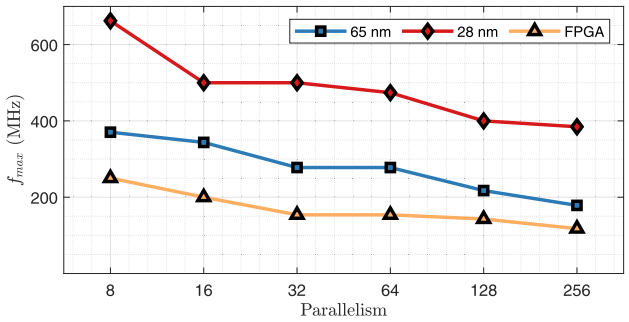


FIGURE 9. The maximum frequency as function of the parallelism degree for different technologies.

where f_{max} is the maximum operating frequency and N_{cycles} is the total number of cycles employed by the decoder. f_{max} depends on the technology (or device) used to synthesize the architecture and the design choices made to implement each unit. The maximum frequency, shown in Fig. 9, is obtained as the reciprocal of $t_{cp,min}$, the minimun critical path delay. N_{cycles} can be computed as the sum of the contributions from the main decoder units in Fig. 6 for both version 1, when (18) applies, and version 2, when (19) applies.

The execution time is shown in Figs. 10 and 11 as a function of the degree of parallelism, for version 1 and version 2, respectively, and for all values of p considered in Table 1. In all cases, the given results are derived from the STM FDSOI 28 nm synthesis.

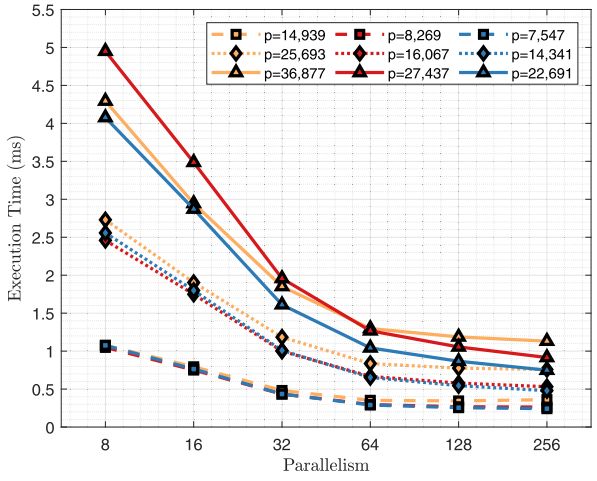


FIGURE 10. The total execution time for version 1 of the decoder.

According to the plots, as expected, by increasing the parallelism, one can progressively reduce the total execution time. However, the achieved speed-up is approximately proportional to the parallelism in the range from 8 to 32, and becomes gradually marginal with higher values. The effectiveness of the parallelism increase is limited by three factors: the increase of the critical path delay, the contribution of non parallelizable units, such as SVbSC, and the

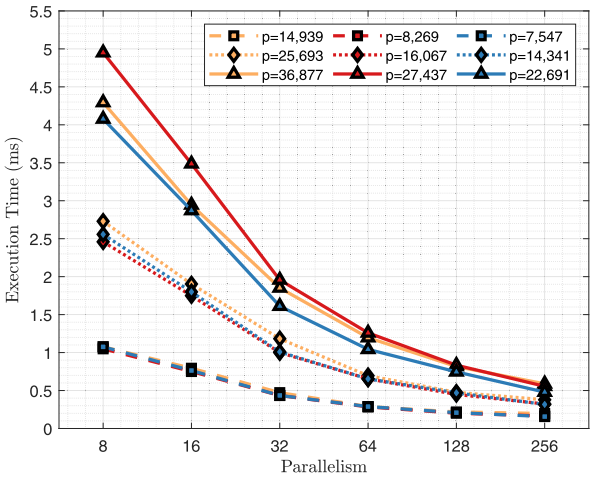


FIGURE 11. The total execution time for version 2 of the decoder.

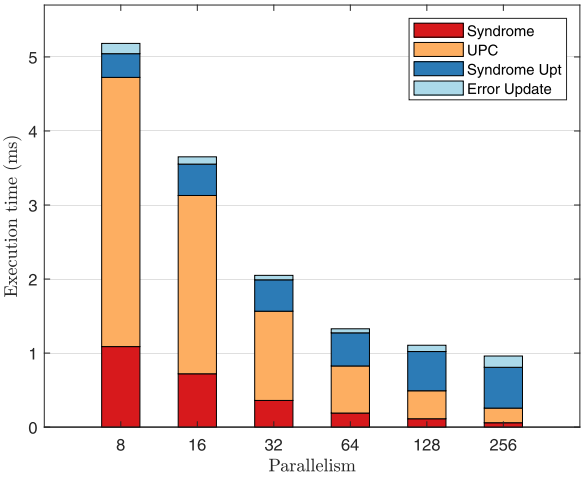


FIGURE 12. Breakdown of the execution time of decoder (version 1) for the most relevant functions, $n_0 = 2$ and $p = 14,939$.

decoding capabilities of the code. The latter is intrinsic to the choice made by the user: for instance, for the case $n_0 = 2$, the number of errors found in a single iteration is on average larger than in the other cases, so requiring more cycles to detect all the errors and then update the syndrome. The limit related to the hardware implementation of the multipliers is mitigated in version 2 of the decoder, which adopts VbSC in the Syndrome Update phase. As a consequence of this modification, the high-parallelism implementations of Fig. 11 are significantly faster than those in Fig. 10. The execution times of the single units are reported, for better evidence, in Figs. 12 and 13 for version 1 and version 2, respectively, assuming $n_0 = 2$ and $p = 14,939$.

We observe, in Fig. 12, that the time required by the Syndrome Update unit increases with the parallelism. This is a consequence of the increase in the critical path delay, t_{cp}. Indeed, the synthesis reports showed that the critical path, both for the ASIC and FPGA implementations, is placed in

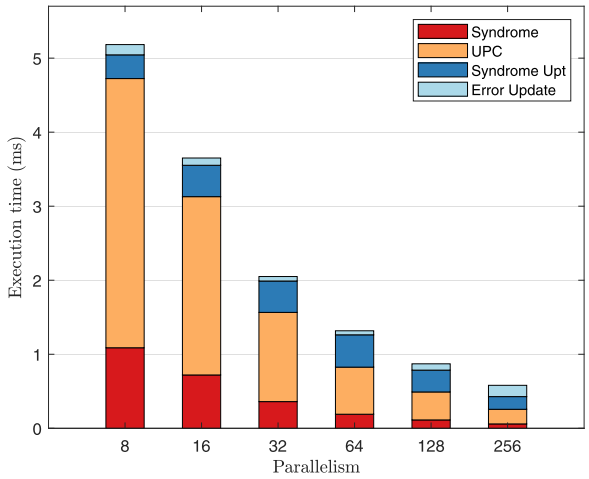


FIGURE 13. Breakdown of the execution time of the decoder (version 2) for the most relevant functions, $n_0 = 2$ and $p = 14,939$.

the VbSC multiplier, along the collapse unit, of the Correlation unit.

Finally, Fig. 13 shows that the Syndrome Update contribution is reduced in version 2, thanks to the different algorithm employed for updating; despite the increase in the critical path, version 2 still benefits from the increase of the parallelism.

B. AREA OCCUPATION AND RESOURCES UTILIZATION

The decoder area occupation depends on the specified parallelism, while it is weakly affected by the parameters in Table 1. Figure 14 shows the area breakdown for the most important units of the decoder.

The figure is referred to version 1, but the area is practically the same for version 2. This is because the difference between the two versions is limited to a small part of the Control unit, thus having a negligible effect on the total area.

According with the implementation presented in [22], the area occupation grows faster than linearly with the parallelism while, in the present work, the increase in the area is almost linear, thanks to the logarithmic structure of the collapse unit.

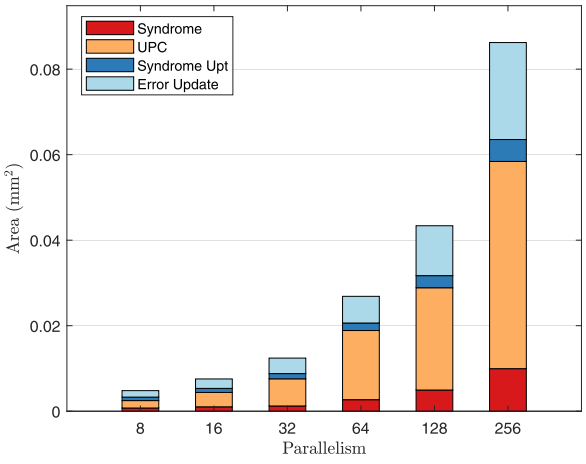


FIGURE 14. The area occupation of the decoder for a 28 nm technology.

The total area occupation as a function of the parallelism is shown in Fig.15 for both the UMC CMOS 65 nm and STM FDSOI 28 nm technologies. The results are referred to a single line of Table 1, with $p = 14,939$. Independently of the technology, the increase in the area is almost linear with the parallelism.

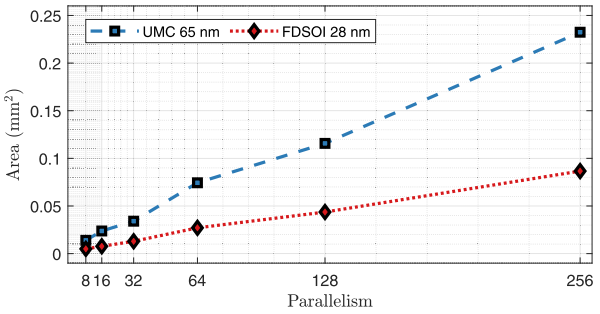


FIGURE 15. Total area occupation of the decoder as a function of the parallelism.

Finally, Fig. 16 shows the percentage of resource utilization for the FPGA implementation. The target FPGA is the Artix-200 platform; this device has 133, 800 LUTs, 367, 600 Flip-Flops (FFs) and 365 Block Random Access Memories (BRAMs). The number of occupied LUTs and FFs tends to increase linearly with the parallelism. However, the number of used BRAMs grows in a less regular way: in the small parallelism range, the number of BRAMs is constant because the size of a single BRAM is enough. On the contrary, in the large parallelism range, the number of required BRAMs grows almost linearly.

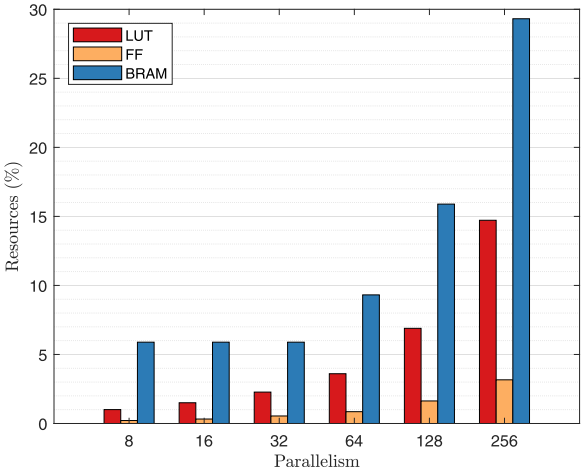


FIGURE 16. Percentage of resource utilization for Artix 7 200.

C. POWER ESTIMATION AND Place&Route RESULTS

In this paper we have not specifically addressed a low power implementation and, consequently, no option to reduce the

power consumption has been adopted. Yet, to complete the characterization of our work, in this section, we provide the power estimation for our implementation. In particular, we have considered both versions of the decoder, assuming the UMC CMOS 65 nm technology with 0.9 V supply voltage and an operating frequency fixed to 100 MHz.

The dissipated power estimate has been carried out with the following procedure: initially the design has been synthesized with Design Compiler®; then, the Verilog netlist has been extracted and simulated with Questa®, using the SDF (Standard Delay Format) delay-based full timing annotation to obtain the switching activity information. Finally, the result has been employed by Design Compiler®to evaluate the dynamic power consumption together with the static power.

With this approach, we have been able to estimate the total power consumption of the design, which we have reported in Figure 17.

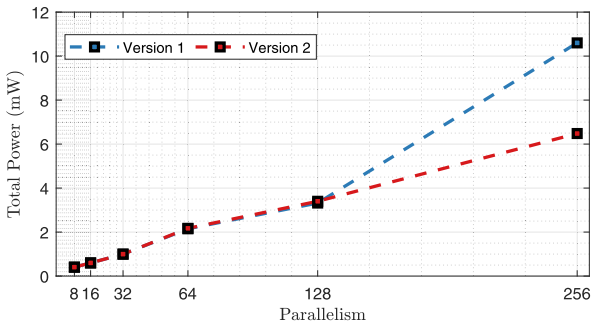


FIGURE 17. Total power consumption for both decoder versions, for the instance with $n_0 = 2$, $p = 14,939$.

The results show that for $n_b = 256$ the architectural choices for version 2 reduce, together with the execution time (as we have already observed in the previous section), also the power consumption.

We have also completed the Place&Route for the UMC CMOS 65 nm; the result, which we have obtained with Cadence®Innovus, starting from the netlist generated by Design Compiler®, is reported in Table 2. As in Figs. 14 and 15, the area is referred to version 1 but, as already mentioned, version 2 has almost the same area occupation. Compared to the pre-Place&Route design, it is possible to verify that the total area increases, as expected, at most by 60%.

TABLE 2. Total Area and Power Consumption of the Chip after Place&Route in mm^2. Results are referred to UMC CMOS 65 nm technology.

	8 bit	16 bit	32 bit	64 bit	128 bit	256 bit
	Area (mm^2) for v1 version					
	0.027	0.039	0.059	0.141	0.209	0.417
	Power (mW)					
v1	0.42	0.68	1.19	2.70	4.35	10.98
v2	0.42	0.68	1.19	2.64	4.23	7.26

Finally, the power consumption for the design generated after the Place&Route has been derived, too. The Verilog netlist and SDF file are generated by Cadence®Innovus, while the input is the Verilog netlist resulting from the synthesis with Design Compiler®. The switching activity is generated with Questa®and then the file is passed to Cadence®Innovus to obtain the power estimate, with the same operating conditions used for the pre-Place&Route case (0.9 V supply voltage and 100 MHz clock frequency). The post-Place&Route total power consumption, reported in Table 2 for both version 1 and version 2, does not differ significantly from the pre-Place&Route estimation.

VI. COMPARISONS

In order to assess the proposed decoder architecture, the obtained results are compared with those appeared in recent literature for two FPGA implementations of LEDAcrypt [19], [21]. The comparison is reported in Table 3, that shows the synthesis results of our LEDAcrypt system for the two target devices used in [19] and [21], i.e., a Spartan-6 device and the Artix-7 c7a200tfbg484. It must be taken into account that [21] considered the Q-decoder only, while in [19], similarly to our analysis, the syndrome computation has been included as well. Comparison has been adapted accordingly, assuming $n_0 = 4$, $p \approx 7,000$, $n_b = 128$ for the Artix-7 case and $n_0 = 2$, $p = 14,939$, $n_b = 32$ for the Spartan-6 case.

TABLE 3. Comparison among LEDAcrypt FPGA implementations.

	Our work		[21]	[19]
	Artix-7 200	Spartan-6	Artix-7 200	Spartan-6
Ex. Time (ms)	0.53	2	0.07	16.1
f_{max} (MHz)	142	120	100	140
LUTs	9,237	2,854	75,962	2,222
FFs	6,003	1,631	32,517	658
BRAMs	58	43	339	13
Ex. Time × LUTs (ms)	4,895	5,708	5,317	35,774

By comparing the two Artix-7 implementations, one can notice that our solution is slower than [21] by a factor 7.5, but much cheaper in terms of occupied FPGA elements, with percentages of saved resources equal to 88%, 82% and 83% for LUTs, FFs and BRAMs, respectively.[3] In order to have a comprehensive figure of merit, we provide in Table 3 the product of the execution time (in ms) by the number of occupied LUTs (Latency × LUTs), which combines both the speed and the hardware complexity achieved by each implementation. This special metric gives very similar values for the two Artix-7 cases, with a relative difference as small as 8%.

[3] In Table 3, the absolute numbers of resources used for the implementation in [21] are estimated from the percentage values given in the paper.

TABLE 4. Comparison between our work and different FPGA implementations of post-quantum cryptography algorithms.

Ref.	Algorithm	Security Level (NIST)	Ex. Time (ms)	Device	Resources			Ex. Time × LUTs (ms)
					LUTs	FF	BRAM	
[38]	BIKE	1	13.02	Artix -7	9,557	3,969	10	124k
			4.57		16,349	4,331	15	74k
			1.9		3,0977	5,092	29	58k
		5	47.76		8,913	3,974	16	425k
			14.81		16,606	4,377	16	245k
			6.11		30,772	5,096	30	188k
[39]	BIKE	1	11.8	Arria 10	51,207			600k
[40]	Classic McEliece	1	47.39	Zynq UltraScale+	77,357	41,338		3.6M
[41]	Classic McEliece	1	0.14	Artix-7	25,327	49,383	168	3.5k
			0.1		39,766	70,453	213	3.9k
			0.09		81,339	132,190	236	7k
		5	0.25		38,669	74,858	303	9k
			0.17		57,134	97,056	349	9k
			0.11	Virtex-7	109,484	168,939	446	12k
[42]	HQC	1	1.46	Artix -7	11,236	7,836	19.5	16k
[43]	SIKE	1	7.4	Artix-7	21,946	24,328	26.5	162k
		2	9.23		24,610	27,759	33.5	227k
		3	14.2		29,447	33,198	39.5	418k
		5	18.4		40,792	49,982	43.5	750k
Our Work	LEDAcrypt	1	2.4	Artix -7	1,043	582	21.5	2.5k
		3	5.8		1,354	716	32	7.8k
		5	11		1,510	785	44.5	11.7k

TABLE 5. Comparison of LEDAcrypt with other cryptosystems.

	Ref.	Device	LUTs	Ex. Time (ms)	Algorithm
FPGA	[44]	Virtex-5	212	11	ECC
	Our work	Artix-7	9,237	0.6	LEDAcrypt
	Ref.	Technology	GE×10^3	Cycles	Algorithm
ASIC	[45]	0.35 μm	135	529,200	RSA
	[46]	65 nm	11	106,700	ECC
	[47]	65 nm	15	351,856	ECC
	[48]	65 nm	11	177,707	ECC
	Our Work	65 nm	165,000	87,110	LEDAcrypt

As a further comment on these results, we notice that the 0.53 ms latency of our design is short enough for the implementation of the LEDAcrypt algorithm on both client and server sides. On the other hand, in our solution the BRAMs usage is significantly reduced; this is because, in our design, only a partial product is addressed during the update by VbSC, while in [21] more partial products are generated in the same iteration.

As for the Spartan-6 designs, we see from the table that the proposed implementation achieves a much shorter latency than [19], at the cost of a slightly larger number of occupied resources; moreover, for our implementation, the Latency × LUTs metric is lower by a factor greater than 6.

This latency improvement is mainly due to the collapse unit that can rotate up to n_b elements in a single cycle, while in [19] the complete rotation of the row is obtained through multiple steps, thus increasing the latency of the whole process.

Another relevant issue concerns the comparison with different designs of PQC schemes, also proposed for the NIST competition. Some relevant examples are shown in Table 4, where we compare the presented architecture against a number of recently published FPGA implementations of well-known PQC algorithms. No ASIC implementations are available for comparisons, for the time being. Details on the algorithms can be found in the references quoted in the first column of the table. All the considered schemes are code-based, except for SIKE that exploits supersingular isogeny graphs. For BIKE and Classic McEliece different implementations have been considered.

The showed comparisons are not entirely fair, as the considered cryptosystems are heterogeneous and not always a direct comparison makes sense. However, the data reported in the table allow for a global overview of the implementation performance and cost of several algorithms proposed in the frame of the NIST competition. From the given figures, it can be seen that LEDAcrypt offers very low latency and complexity with respect to the other cryptosystems. In particular, from the rightmost column in the Table, which gives the Latency × LUTs product, we see that LEDAcrypt

(with $n_b = 8$ and $n_0 = 2$) shows the lowest value of this metric among all considered implementations. Moreover, despite the BIKE algorithm is similar to LEDAcrypt, its implementation turns out to be more expensive than LEDAcrypt in terms of occupied resources. On the whole, the reported results suggest that the proposed implementation is an effective way to realize a post-quantum cryptosystem.

Finally, in Table 5, we give a comparison with a few public-key cryptography schemes that are widely used nowadays, such as ECC and RSA.

Of course, the security level of these systems is not comparable with the security provided by LEDAcrypt and the other PQC systems. However, the purpose of such a table is to have a first evaluation of the additional latency and complexity of a post-quantum cryptosystem, like LEDAcrypt, with respect to quantum-vulnerable schemes which are currently in use.

In the first part of the table, we compare our architecture mapped on an Artix-7 device, for the case $p = 14,939$ and $n_b = 128$, against the Virtex-4 implementation proposed in [44]. The second part of the table presents instead the comparison between our UMC CMOS 65 nm implementation and four ASIC designs: one for RSA [45] and three for ECC [46]–[48]; a large number of additional ECC implementations are reported in [49], but the comparison with them is here omitted for the sake of brevity. As for the ASIC case, we notice that the difference between our post-quantum cryptosystem (with $p = 14,939$ and $n_b = 128$) and the other ones is limited in terms of latency: in particular, the LEDAcrypt ASIC implementation is faster than the considered ECC designs by a factor ranging from 1.2 to 4. The ASIC implementation of LEDAcrypt is also much faster than the reported RSA design. On the contrary, the differences in terms of equivalent gates result to be much larger: while LEDAcrypt needs $165 \cdot 10^6$ equivalent gates, the complexity of the considered ECC implementations is in the order of $10 \cdot 10^3$ equivalent gates. A slightly higher complexity is required for the ASIC RSA design. The increase in the number of gates, however, is quite obvious and expected, and must be interpreted as the price to pay for designing cryptosystems able to resist against quantum computers.

VII. CONCLUSION

This work has introduced significant improvements with respect to previous implementations of the recently proposed LEDAcrypt code-based post-quantum cryptosystem. In particular, we have presented an ASIC implementation, which was missing in existing literature, to the best of our knowledge. Moreover, as regards the FPGA implementation, already studied in the literature, our architecture is characterized by an excellent trade-off between execution time and area occupation. In particular, by assuming the Latency $\times$ LUTs product as a figure of merit, we have shown that our design is able to significantly reduce it, up to a factor of 6 for the Spartan-6 device. Moreover, our implementation compares favorably with other code-based schemes proposed for PQC and is even faster than known realizations of classical public-key cryptography schemes like ECC or RSA.

Our focus has been on the implementation of a KEM with ephemeral keys and assuming the parameters chosen for the second round of the NIST PQC competition. However, the design can be extended to different scenarios, as well as easily scaled for taking into account parameters updating, as probably required in future versions of the algorithm.

REFERENCES

[1] M. Giles, *IBM New 53 Qubit Quantum Computer is the Most Powerful Machine You Can Use*. Cambridge, MA, USA: MIT Press, 2018.
[2] J. Hsu, "CES 2018: Intel's 49-qubit chip shoots for quantum supremacy," *IEEE Spectr.*, to be published.
[3] J. Porter, "Google confirms 'quantum supremacy' breakthrough," VERGE, 2019.
[4] F. Arute, K. Arya, R. Babbush, D. Bacon, and J. C. Bardin, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
[5] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, Santa Fe, NM, USA, 1994, pp. 124–134.
[6] National Institute of Standards and Technology. *Post-Quantum Crypto Project*. Accessed: 2020. [Online]. Available: http://csrc.nist.gov/groups/ST/post-quantum-crypto/
[7] E. Berlekamp, R. McEliece, and H. van Tilborg, "On the inherent intractability of certain coding problems (Corresp.)," *IEEE Trans. Inf. Theory*, vol. 24, no. 3, pp. 384–386, May 1978.
[8] G. Alagic, J. Alperin-Sheriff, D. C. Apon, D. A. Cooper, Q. H. Dang, J. M. Kelsey, Y.-K. Liu, C. A. Miller, D. Moody, R. C. Peralta, R. A. Perlner, A. Y. Robinson, and D. C. Smith-Tone. (Jul. 2020). *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8309.pdf
[9] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," *Deep Space Netw. Prog. Rep.*, vol. 44, pp. 114–116, Jan. 1978.
[10] M. Baldi, F. Chiaraluce, and M. Bianchi, "Security and complexity of the McEliece cryptosystem based on quasi-cyclic low-density parity-check codes," *IET Inf. Secur.*, vol. 7, no. 3, pp. 212–220, 2013.
[11] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini. *LEDAkem and LEDApkc Website*. Accessed: 2020. [Online]. Available: https://www.ledacrypt.org/
[12] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Ghosh, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. Zémor. *BIKE Website*. https://bikesuite.org/
[13] N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J. Bos, J.-C. Deneuville, P. Gaborit, C. A. Melchor, E. Persichetti, J.-M. Robert, P. Véron, and G. Zémor. *HQC Website*. https://https://pqc-hqc.org/.
[14] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.
[15] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, 2001.
[16] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "LEDAcrypt: QC-LDPC code-based cryptosystems with bounded decryption failure rate," in *Code-Based Cryptography*, M. Baldi, E. Persichetti, and P. Santini, Eds. Cham, Switzerland: Springer, 2019, pp. 11–43.
[17] D. Apon, R. Perlner, A. Robinson, and P. Santini, "Cryptanalysis of LEDAcrypt," in *Advances in Cryptology–(CRYPTO)*, D. Micciancio and T. Ristenpart, Eds. Cham, Switzerland: Springer, 2020, pp. 389–418.
[18] P. Santini, M. Battaglioni, M. Baldi, and F. Chiaraluce, "Analysis of the error correction capability of LDPC and MDPC codes under parallel bit-flipping decoding and application to cryptography," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 4648–4660, Aug. 2020.
[19] J. Hu, M. Baldi, P. Santini, N. Zeng, S. Ling, and H. Wang, "Lightweight key encapsulation using LDPC codes on FPGAs," *IEEE Trans. Comput.*, vol. 69, no. 3, pp. 327–341, Mar. 2020.
[20] B. Bui. *Hardware Implementation of Q-Decoding Algorithm in LEDAkem Encapsulation Mechanism*. [Online]. Available: https://people-ece.vse.gmu.edu/coursewebpages/ECE/ECE646/F19/project.html

[21] D. Zoni, A. Galimberti, and W. Fornaciari, "Efficient and scalable FPGA-oriented design of QC-LDPC bit-flipping decoders for post-quantum cryptography," *IEEE Access*, vol. 8, pp. 163419–163433, 2020.

[22] K. Koleci, M. Baldi, M. Martina, and G. Masera, "A hardware implementation for code-based post-quantum asymmetric cryptography," in *Proc. 3rd Italian Conf. Cybersecurity (ITASEC)*, vol. 2597, Ancona, Italy, Feb. 2020, pp. 141–152.

[23] D. Zoni, A. Galimberti, and W. Fornaciari, "Flexible and scalable FPGA-oriented design of multipliers for large binary polynomials," *IEEE Access*, vol. 8, pp. 75809–75821, 2020.

[24] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *Problems Control Inf. Theory*, vol. 15, no. 2, pp. 159–166, 1986.

[25] 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*, R. Ostrovsky, R. De Prisco, and I. Visconti, Eds. Berlin, Heidelberg: Springer, 2008, pp. 246–262.

[26] 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 Int. Symp. Inf. Theory*, Istanbul, Turkey, Jul. 2013, pp. 2069–2073.

[27] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "LEDAkem: A post-quantum key encapsulation mechanism based on QC-LDPC codes," in *Post-Quantum Cryptography*, T. Lange and R. Steinwandt, Eds. Cham, Switzerland: Springer, 2018, pp. 3–24.

[28] N. Drucker, S. Gueron, and D. Kostic, "QC-MDPC decoders with several shades of gray," in *Post-Quantum Cryptography*, J. Ding and J.-P. Tillich, Eds. Cham, Switzerland: Springer, 2020, pp. 35–50.

[29] N. Drucker, S. Gueron, and D. Kostic, "On constant-time QC-MDPC decoders with negligible failure rate," in *Code-Based Cryptography*, M. Baldi, E. Persichetti, and P. Santini, Eds. Cham, Switzerland: Springer International Publishing, 2020, pp. 50–79.

[30] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "A failure rate model of bit-flipping decoders for QC-LDPC and QC-MDPC code-based cryptosystems," in *Proc. 17th Int. Joint Conf. e-Business Telecommun.*, vol. 3, Paris, France, 2020, pp. 238–249.

[31] N. Sendrier and V. Vasseur, "About low DFR for QC-MDPC decoding," in *Post-Quantum Cryptography*, J. Ding and J.-P. Tillich, Eds. Cham, Switzerland: Springer, 2020, pp. 20–34.

[32] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers," *Doklady Akademii Nauk SSSR*, vol. 145, no. 2, pp. 293–294, 1962.

[33] A. Schönhage and V. Strassen, "Schnelle multiplikation grosser zahlen," *Computing*, vol. 7, nos. 3–4, pp. 281–292, 1971.

[34] K. Millar, M. Łukowiak, and S. Radziszowski, "Design of a flexible Schönhage-Strassen FFT polynomial multiplier with high-level synthesis to accelerate HE in the cloud," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Cancun, Mexico, Dec. 2019, pp. 1–5.

[35] X. Feng, S. Li, and S. Xu, "RLWE-oriented high-speed polynomial multiplier utilizing multi-lane stockham NTT algorithm," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 3, pp. 556–559, Mar. 2020.

[36] C. Rafferty, M. O'Neill, and N. Hanley, "Evaluation of large integer multiplication methods on hardware," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1369–1382, Aug. 2017.

[37] T. Chou, "QcBits: Constant-time small-key code-based cryptography," in *Cryptographic Hardware and Embedded Systems–(CHES)*, B. Gierlichs and A. Y. Poschmann, Eds. Berlin, Germany: Springer, 2016, pp. 280–300.

[38] J. Richter-Brockmann, J. Mono, and T. Güneysu, "Folding BIKE: Scalable hardware implementation for reconfigurable devices," Cryptol. ePrint Arch., Tech. Rep. 2020/897, 2020. [Online]. Available: https://eprint.iacr.org/2020/897

[39] A. H. Reinders, R. Misoczki, S. Ghosh, and M. R. Sastry, "Efficient BIKE hardware design with constant-time decoder," in *Proc. IEEE Int. Conf. Quantum Comput. Eng. (QCE)*, Denver, CO, USA, Oct. 2020, pp. 197–204.

[40] M. López-García and E. Cantó-Navarro, "Hardware-software implementation of a McEliece cryptosystem for post-quantum cryptography," in *Advances in Information and Communication*, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham, Switzerland: Springer, 2020, pp. 814–825.

[41] M. Albrecht, D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. L. V. Maram, I. von Maurich, R. Misoczki, R. Niederhagen, K. Paterson, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, C. J. Tjhai, M. Tomlinson, and W. Wang. (2020). *Classic McEliece*. [Online]. Available: https://classic.mceliece.org/hardware.html

[42] C. Aguilar Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J. Bos, J. Deneuville, A. Dion, P. Gaborit, J. Lacan, E. Persichetti, J. Robert, P. Véron, and G. Zemor. (2020). *Hamming Quasi-Cyclic (HQC)–Third Round Version–Updated Version 10/01/2020*. p. 47. [Online]. Available: http://pqc-hqc.org/doc/hqc-specification_2020-10-01.pdf

[43] B. Koziel, A. Ackie, R. El Khatib, R. Azarderakhsh, and M. M. Kermani, "SIKE'd up: Fast hardware architectures for supersingular isogeny key encapsulation," *IEEE Trans. Circuits Syst. I, Regular Papers*, vol. 67, no. 12, pp. 4842–4854, Dec. 2020.

[44] D. B. Roy, P. Das, and D. Mukhopadhyay, "ECC on your fingertips: A single instruction approach for lightweight ECC design in GF(p)," in *Proc. Int. Conf. Sel. Areas Cryptogr.* Sackville, NB, Canada: Springer, 2015, pp. 161–177.

[45] S. Yesil, A. N. Ismailoglu, Y. C. Tekmen, and M. Askar, "Two fast RSA implementations using high-radix montgomery algorithm," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, Vancouver, BC, Canada, May 2004, pp. 557–560.

[46] R. Azarderakhsh, K. U. Jarvinen, and M. Mozaffari-Kermani, "Efficient algorithm and architecture for elliptic curve cryptography for extremely constrained secure applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 4, pp. 1144–1155, Apr. 2014.

[47] B. Koziel, R. Azarderakhsh, and M. Mozaffari-Kermani, "Low-resource and fast binary edwards curves cryptography," in *Proc. Int. Conf. Cryptol. India*. New Delhi, India: Springer, 2015, pp. 347–369.

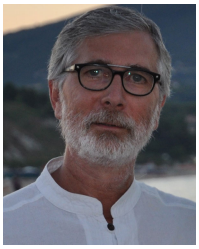
[48] S. S. Roy, K. Järvinen, and I. Verbauwhede, "Lightweight coprocessor for Koblitz curves: 283-bit ECC including scalar conversion with only 4300 gates," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Saint Malo, France: Springer, 2015, pp. 102–122.

[49] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Elliptic curve lightweight cryptography: A survey," *IEEE Access*, vol. 6, pp. 72514–72550, 2018.

KRISTJANE KOLECI (Graduate Student Member, IEEE) received the M.S. degree in electronic engineering from the Politecnico di Torino, Italy, in 2019, where she is currently pursuing the Ph.D. degree with the VLSI-Laboratory Group. She is also the Vice-Chair of the IEEE Women in Engineering Student Branch Affinity Group with the Politecnico di Torino. Her research interests include the design hardware architectures of algorithms for post-quantum cryptography and error correcting codes.

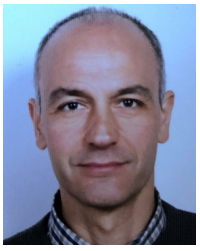
PAOLO SANTINI (Member, IEEE) received the master's degree (Hons.) in electronic engineering and the Ph.D. degree in electronic, computer and telecommunications engineering from the Universita Politecnica delle Marche, in 2016 and 2020, respectively. From November 2019 to February 2020, he has been a Research Associate with Florida Atlantic University. He is currently working as a Postdoctoral Research Fellow with the Universita Politecnica delle Marche. He has participated to the NIST PQC standardization process as a team member of LEDAcrypt. His research interests include coding theory, security, and cryptography.

MARCO BALDI (Senior Member, IEEE) is currently an Associate Professor of telecommunications with the Department of Information Engineering, Polytechnic University of Marche Ancona, Italy, where he also coordinates the local node of the CINI Cybersecurity National Laboratory and takes part in the Research and Service Center for Privacy and Cybersecurity (CRiSPY). He is the coauthor of over 150 scientific articles and one book, and holds four patents. His research interests include cybersecurity, with special interest in coding and cryptography for information security and reliability. His research activity is carried out in collaboration with national and international companies and institutions. He has received numerous awards for research activities. He has given numerous invited speeches and seminars at international conferences and international research institutions. He is a member of AEIT, EURACON, CINI, CNIT, GTTI, the IEEE Communications Society, and the IEEE Information Theory Society. He also serves as a Senior Associate Editor for IEEE COMMUNICATIONS LETTERS and as an Associate Editor for the *EURASIP Journal on Wireless Communications and Networking* and the *Information* journal (MDPI).



FRANCO CHIARALUCE (Senior Member, IEEE) was born in Ancona, Italy, in 1960. He received the Laurea degree *(summa cum laude)* in electronic engineering from the Università di Ancona, in 1985. Since 1987, he has been with the Department of Electronics and Automatics, Università di Ancona. He is currently a Full Professor of telecommunications with the Università Politecnica delle Marche, Ancona, where he is also the Coordinator of the Ph.D. course in information engineering. He has coauthored more than 300 scientific articles and three books, and holds three patents. On his research topics, he collaborates with national and international companies. His main research interests include communication systems theory and design, with a special emphasis on error correcting codes, cryptography, and physical layer security. He is also a member of the IEICE.

MAURIZIO MARTINA (Senior Member, IEEE) received the M.S. and Ph.D. degrees in electronic engineering from the Politecnico di Torino, Italy, in 2000 and 2004, respectively. He is currently an Associate Professor with the VLSI-Laboratory Group, Politecnico di Torino. He is the Counselor of the IEEE Student Branch, Politecnico di Torino. He has more than 100 scientific publications. His research interests include VLSI design and implementation of architectures for digital signal processing, video coding, communications, artificial intelligence, machine learning, and event-based processing. He is a Professional Member of IEEE HKN. He is an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS–I: REGULAR PAPERS.



GUIDO MASERA (Senior Member, IEEE) received the Dr.-Ing. *(summa cum laude)* and Ph.D. degrees in electronic engineering from the Politecnico di Torino, Italy. Since 1992, he has been a Professor with the Electronic Department, Politecnico di Torino. He has more than 200 publications, two patents, and was a designer of several ASIC components. His research interests include several aspects in the design of digital integrated circuits and systems, with a special emphasis on high-performance architectures for communications, forward error correction, image and video coding, cryptography, and hardware accelerators for machine learning. He is an Associate Editor of *Electronics* MDPI and a former Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-I: REGULAR PAPERS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-II: EXPRESS BRIEFS, and the *IET Circuits, Devices & Systems*.