Quantum-Proof Lightweight McEliece Cryptosystem Co-processor Design

Rashmi Agrawal*, Lake Bu†, Michel A. Kinsy‡

* *Adaptive and Secure Computing Systems (ASCS) Laboratory*
Department of Electrical and Computer Engineering, Boston University
Boston, MA
†*The Charles Stark Draper Laboratory, Inc.*
Cambridge, MA
‡ *Adaptive and Secure Computing Systems (ASCS) Laboratory*
Department of Electrical and Computer Engineering, Texas A&M University
College Station, TX

***Abstract*—Due to the rapid advances in the development of quantum computers and their susceptibility to errors, there is a renewed interest in error correction algorithms. In particular, error correcting code-based cryptosystems have reemerged as a highly desirable coding technique. This is due to the fact that most classical asymmetric cryptosystems will fail in the quantum computing era. However, code-based cryptosystems are still secure against quantum computers, since the decoding of linear codes remains NP-hard even on these computing systems. One such code-based cryptosystem was proposed by McEliece. The classic McEliece cryptosystem uses binary Goppa code, which is known for its good code rate and error correction capability. However, its key generation and decoding procedures have a high computation complexity. In this work, we propose the design of a public-key encryption and decryption co-processor based on a new variant of the McEliece cryptosystem. This co-processor takes advantage of non-binary Orthogonal Latin Square Code to achieve much smaller computation complexity and key size. We also propose a hardware-cost efficient, fully-parameterized FPGA-based implementation of the co-processor to perform fast encoding and decoding operations. When compared to an existing classic McEliece cryptosystem, we observe a speed up of about $3.3\times$.**

***Index Terms*—Code-based post-quantum cryptosystem, McEliece public-key encryption, Orthogonal Latin Square Codes.**

I. Introduction

Since the possibility of using quantum effects in computation was brought up by Feynman in 1959, numerous efforts have been dedicated to realize and even commercialize quantum computers. In the past three years, a number of significant milestones have been reached in this area. From late 2017 to early 2018, technology companies such as IBM [1], Intel [2], and Google [3], announced their construction and testing of 50-, 49-, and 72-qubit computers, respectively. In July 2018, for the first time, researchers at the University of Sydney successfully realized a multi-qubit computation on a system of trapped ions, which is believed to be the leading platform for building general quantum computers [4]. In December 2018, IonQ claimed to have built a quantum computer with 160 qubits. Besides these advances in the physical implementation of quantum computers, key breakthroughs in the verification of quantum computation were also achieved [5], and more efficient error correction schemes were recently proposed as well.

These efforts are rooted in the fact that quantum computers promise greater computational power. But these developments also bring with them burning security concerns. For example, Shor's algorithm [6], leveraging quantum Fourier transforms, is able to solve the integer factorization problem efficiently. Therefore, current popular cryptographic algorithms such as RSA, ElGamal, Diffie-Hellman, and ECC, which rely on the hardness of integer factorization and discrete logarithm (the two are also closely related), are vulnerable to quantum computer-based algorithms [7]–[9]. Moreover, for symmetric cryptography, Grover's algorithm [10] applies fast search in the key space, so that the security level of symmetric encryptions can be reduced to half of its original - e.g., the 128-AES now provides only 64-bit security, failing to meet the 112-bit minimum security level recommended by the National Institute of Standards and Technology (NIST).

In response to the aforementioned security challenges associated with quantum computers, a number of new cryptosystems have been proposed for the post-quantum era. In early 2017, NIST launched a campaign for a post-quantum cryptography standard. In February 2019, totally 27 out of 69 candidates for this new standard made it to the second round of competition [11]. Among the 27, the two most likely contenders are the lattice-based cryptosystems (12 candidates), and the code-based cryptosystems (8 candidates). Both of them are able to construct public-key cryptosystems and key exchange mechanisms. Compared to the popular lattice-based ring-learning with error (Ring-LWE) cryptosystem, error-correcting code (ECC)-based schemes have a much larger key size, which is considered a drawback. However, they do boast the advantage of withstanding the test of time. For example, since its formulation by Robert McEliece in 1978 [12], the McEliece code-based technique has so far proven to be cryptanalysis resistant (although sometimes increasing the key size is necessary to meet a desired security level).

The conventional McEliece cryptosystem uses binary Goppa code, which has good code rate and error correction capability. However, compared to other binary codes, the encoding and decoding (error-correction) of binary Goppa codes have relatively high complexity. This is because they involve intensive computations over finite fields, including modulo polynomial operations. Moreover, the key size of McEliece systems is usually large. For a binary Goppa code with a $k \times n$ generating matrix, the key size is kn with k, n in thousands of bits. Therefore, to address these issues, we propose the design of a new variant of the McEliece cryptosystem and its encryption-decryption co-processor. The proposed system is based on the generalized non-binary Orthogonal Latin Square Code (OLSC), which is known for its simple encoding and decoding algorithms, leading to an efficient hardware implementation. In addition, the non-binary OLSC is able to work with non-binary messages through binary matrices. Hence, a long message can be processed using relatively small matrices, reducing the key size significantly. Major contributions of this work are two-fold:

1. **Design:** McEliece cryptosystem encryption-decryption co-processor design based on non-binary Orthogonal Latin Square Code.

2576-6996/20/$31.00 ©2020 IEEE
DOI 10.1109/ICCD50377.2020.00029

2. **Implementation:** A fully optimized and parameterized FPGA-based implementation of the proposed co-processor design to perform fast operations.

To the best of our knowledge, this work is the first proposal of McEliece cryptosystem based on Orthogonal Latin Square Codes, and so is its hardware implementation.

Rest of the paper is organized as follows. In Section II, we discuss the existing hardware implementations of the classic McEliece variant. Section III provides a brief background of the McEliece cryptosystem and OLSC. In Section IV, we discuss the proposed architecture design along with its implementation. And then, in Sections V, VI, and VII, we discuss the complexity, evaluate the performance, and present a brief security analysis of our implementation.

II. Related Work

While there are no existing implementations for an OLSC-based McEliece public-key encryption scheme, there exist a few implementations for the classic variant. Eisenbarth et al. [13] presented MicroEliece, a McEliece encryption scheme implementation on a Xilinx Spartan-3AN FPGA and on a low-cost 8-bit AVR microprocessor. The parameters chosen in their implementation provide 80 bits mid-term security with a public key size of 3,502 Kb. Their implementation is not parameterized and consists of only the encryption and decryption modules, leaving out the key generation operation. Moreover, the authors implement only the polynomial arithmetic operations that were necessary for solving main equations.

Ghosh et al. [14] present an embedded co-processor for the McEliece cryptosystem. The proposed design is implemented using the GEZEL hardware description language. Their implementation is based on a software-hardware co-design approach, in which crucial parts of the algorithm are implemented on a Spartan-3AN FPGA, while a PicoBlace microcontroller is programmed to carry out the required vector and matrix arithmetic operations. The authors claim to accelerate both the encryption and decryption operations using this embedded co-processor; however, they only describe the hardware cost and latency associated with the decryption module.

Massolino et al. [15] also present a hardware implementation on Spartan-3AN for the binary Goppa code based McEliece cryptosystem. In their work, the authors discuss only the decryption unit's hardware design implementation along with its hardware cost and latency. Shoufan et al. [16] present a full CCA2 secure design implementation of the McEliece cryptosystem based on binary Goppa code. Their parameters provide 103 bit security, which is still lower than the minimum security level of 112-bits required by NIST. Even though the authors provide a very detailed description of their hardware design, they do not discuss the results in detail. Moreover, the authors do not provide a break down of the hardware cost for the key generation, encryption, and decryption modules. Another drawback in their design is its non-constant time implementation, making it susceptible to side-channel attacks and other potential security flaws. We compare the performance and hardware cost of the proposed co-processor to these implementations in Section VI.

III. The OLSC-based Cryptosystem

In this section, we will briefly introduce the McEliece cryptosystem algorithms followed by a preliminary view of the OLSC that replace the binary Goppa code.

A. The McEliece Algorithm

The detailed protocol of the public-key cryptosystem (PKC) can be found in [12]. Here, we provide a brief introduction to aid the presentation of the co-processor.

Key generation(KeyGen): Alice picks a binary (n, k, t) ECC code C with k information (plaintext) bits, n total codeword length, and the capability of correcting up to t random errors. The $k \times n$ encoding matrix of C is denoted by G. Alice also picks a $k \times k$ binary non-singular matrix S and a $n \times n$ permutation matrix P. Then Alice computes the public key G' as:

$$G' = SGP \tag{1}$$

Alice keeps S, G, P as the private key.

Encryption(Enc): Bob converts his message (plaintext) into a k-bit binary vector m, and generates the n-bit cipher $\{c\}$ as:

$$c = mG' + e, \tag{2}$$

where e is a binary vector weight t.

Decryption(Dec): Alice decrypts the cipher by performing:

$$m = (\text{Decode}(cP^{-1}))S^{-1} \tag{3}$$

where Decode() stands for the error correction function of C, and P^{-1}, S^{-1} are the inverse matrices of P, S respectively.

B. The Orthogonal Latin Square Code (OLSC):

The binary Goppa code used in the classic McEliece cryptosystem has an algebraic structure that increases the decoding complexity and leads to slow decoding in the cryptosystems based on these codes. This is the reason for replacing the binary Goppa code with the OLSC, which has no algebraic structure and exhibits simple combinatorial properties. The OLS-based code, a new class of multiple-error correcting code, was introduced by Hsiao et al. [17]. The codes are obtained from a set of mutually orthogonal Latin squares by systematically adding high redundancy to the decoding matrix. High speed and simple decoding is achieved through this additional redundancy.

The OLSC technique is a t-error-correcting binary code with a decoding matrix:

$$H = \begin{bmatrix} M & | & I_{2tq} \end{bmatrix}$$

where M consists of submatrices $M_1, \cdots, M_{2t}$, and I is an identity matrix of order $2tq$. The submatrix M_1 is a diagonal matrix, with q $1s$ in the diagonal:

$$M_1 = \begin{bmatrix} 11\cdots1 & & & & 0 \\ & 11\cdots1 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ 0 & & & & 11\cdots1 \end{bmatrix}_{q \times q^2}$$

The submatrix M_2 consists of q $I_{q\times q}$ identity matrices in the form:

$$M_2 = \begin{bmatrix} I_q & I_q & \cdots & I_q \end{bmatrix}_{q \times q^2}$$

and $M_3, \cdots, M_{2t}$ consists of $2(t-1)$ orthogonal Latin squares sized $q \times q$. The set of Latin squares can be denoted by $L \in \{1, 2, \cdots, q-1\}$. Of the total $q-1$ orthogonal Latin squares, only $2t-2$ Latin squares are to be chosen at random to be a part of the matrix M. The set of these Latin squares can be denoted as

$$L_1 = \begin{bmatrix} l_{ij}^1 \end{bmatrix}_{q \times q}$$

$$L_2 = \begin{bmatrix} l_{ij}^2 \end{bmatrix}_{q \times q}$$

$$\vdots$$

$$L_{2t-2} = \begin{bmatrix} l_{ij}^{2t-2} \end{bmatrix}_{q \times q}$$

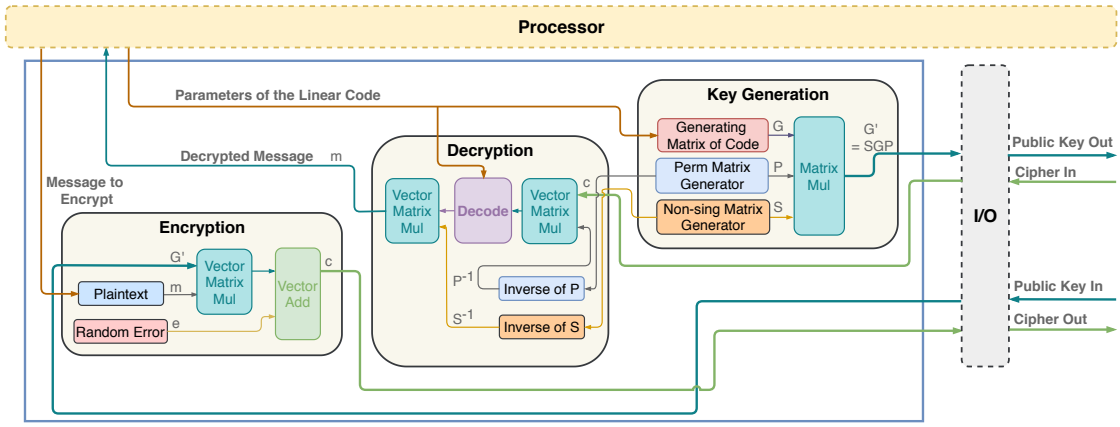

Fig. 1. McEliece Cryptosystem Co-Processor Architecture.

The encoding matrix for OLS codes is $G = [I|M^\top]$, where $^\top$ stands for transpose. While constructing the encoding matrix, the way orthogonal Latin squares are placed in the encoding matrix can be permuted to create different OLSCs with the same parameters. The t-error-correcting codes thus generated will have q^2 data bits and $2tq$ check bits per word.

The decryption along with the decoding procedure of OLSC is presented in Algorithm 1, which consists of mostly binary linear operations. Thus, it can be carried out quickly and very efficiently in hardware. By replacing the binary Goppa code with the non-binary OLSC, the decoding stage only requires (i) binary vector-matrix multiplication and (ii) k parallel majority votings among q non-binary vectors. This feature enables much faster decryption time than the Goppa-code based scheme, as shown in Table I.

Algorithm 1. Decryption in OLSC-based McEliece Cryptosystem

Let $G' = SGP$ and t be the public key, and $\{G, S, P\}$ the private key, where G is a $k \times n$ OLSC encoding matrix with random permutation of columns, and H as its corresponding decoding matrix. Let each Latin square be of size $q \times q$, m be the plaintext, and c the encrypted cipher.

Precompute: S^{-1}, P^{-1} as the inverse to S, P.

$c' \leftarrow cP^{-1}$
$u \leftarrow Hc' \times H$
for i=0 **to** n
    $m'_i \leftarrow (u_i > q/2)? \sim c'_i : c'_i$
$m \leftarrow m'S^{-1}$
return m

TABLE I
COMPLEXITY ANALYSIS OF DECODING

	Finite Field Ops	Latency (cycles)
Binary Goppa Code-based	$O(n^2)$	$O(n)$
OLSC-based	0	$O(1)$

Thanks to this much simpler decoding mechanism, in the OLSC-based McEliece cryptosystem co-processor, we are able to design a single-cycle (one-step) decoding unit. This design is much faster than an equivalent binary Goppa code-based system. The non-binary OLSC is able to encode a kb-bit plaintext with a $k \times n$ generating matrix G (b being the size of each non-binary symbol), while binary Goppa code can only deal with k-bit plaintexts with such a matrix.

To summarize, given the same size of plaintext, the key size of the non-binary OLSC-based McEliece cryptosystem is $1/b$ that of the Goppa code-based version. Given a proper verification on the security level of the OLSC-based scheme with various decoding techniques, it could serve as an efficient and high speed variant of the McEliece cryptosystem. Such a proof is outside the scope of this paper, and thus we leave it for future work.

IV. CRYPTOSYSTEM CO-PROCESSOR ARCHITECTURE

The McEliece public key encryption cryptosystem co-processor has three major modules as shown in Fig 1: *Key Generation*, *Encryption* and *Decryption*. Of the three modules, the KeyGen unit has the highest complexity and, therefore, most existing hardware implementations perform KeyGen in software. The decryption unit has the second highest complexity, and it consumes the most hardware resources, especially its *Decode* stage. Hence, our efficient implementation primarily targets the *Decode* sub-module.

A. FPGA-based Implementation

As discussed in Section III, the implementation of the co-processor requires large matrix-multiplication and matrix-vector operations. Thus, the challenge is to optimize the implementation so as to maximize the performance and accuracy, while keeping the hardware cost as low as possible. The architecture consists of three core building blocks. Each of these building blocks represents an equation in the main algorithm. Figure 1 shows the overall system architecture of the co-processor using the commonly-shared hardware modules. The KeyGen module computes Eq. (1), which breaks down to two matrix multiplication operations. The Enc module computes Eq. (2) and consists of matrix-vector multiplication and vector addition operations. Similarly, the Dec module computes Eq. (3) and consists of two matrix-vector multiplication operations along with a decoding operation. So, the implementation of the McEliece cryptosystem co-processor reduces to constructing modules for performing these basic operations.

The main advantage of the proposed design scheme becomes visible via the hardware implementation. Encoding and decoding, for Goppa codes, involve long polynomial division and solving equations

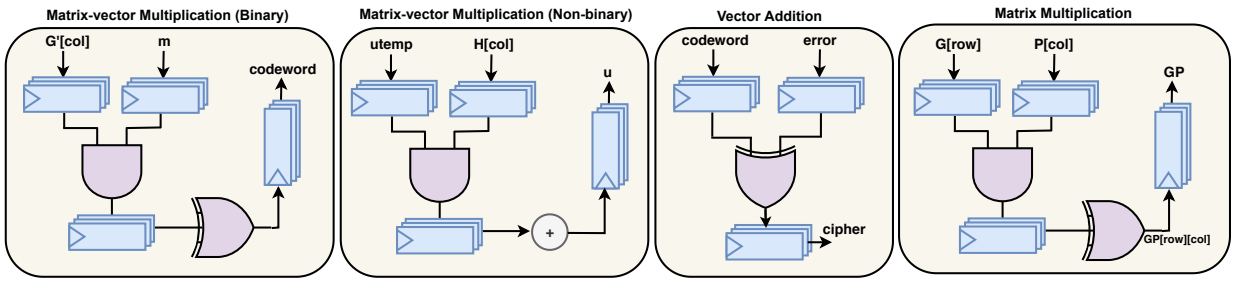

Fig. 2. All basic operations in the McEliece cryptosystem algorithm: *Matrix-vector Multiplication (Binary)*, *Matrix-vector Multiplication (Non-binary)*, *Vector Addition*, and *Matrix Multiplication.*

over finite fields, which can be expensive to implement in hardware. For orthogonal Latin square codes, we just need to perform the binary matrix and vector operations, which leads to an inexpensive hardware implementation. Furthermore, we precompute and store the non-singular matrix S and its inverse, the permutation matrix P and its inverse, and the encoding matrix G which reduces the hardware cost significantly. We would like to highlight, though, that if a different key needs to be generated, then all these matrices will have to be regenerated as well. The amount of memory required to store the matrices will depend on the chosen parameters for the co-processor.

B. Modules and Design Choices

The basic operations in our algorithm are matrix-vector multiplication, matrix multiplication, vector addition, and error correction. We will discuss the design implementation for each of these submodules next.

1) ***Matrix-vector Multiplication****:* Matrix-vector multiplication is the most frequently used submodule as it is required in both the encryption and decryption modules. However, matrix-vector multiplication will be required in two variants i.e. binary and non-binary versions. The encryption module requires only the binary version, whereas the decryption module requires both versions. Figure 2 shows that in matrix-vector multiplication (binary) the elements of the matrix are accessed in column-major order and the message vector is treated as a row. So we have access to all the elements of a column from a matrix at once and also all the elements of the vector. Next, we perform element-wise multiplication using the AND operation, and the summation of the multiplication is performed using unary XOR operation on the result of the multiplication. This is possible because both the matrix and the vector are binary and the resultant vector elements are required to be binary. Thus, we reduce the hardware cost significantly by performing multiplication and addition operations without using any multipliers and adders.

Matrix-vector multiplication (non-binary) also shown in Figure 2, differs from the binary variant in the summation stage. As we will need to compare integer values in majority voting, the non-binary variant of matrix-vector multiplication is required specifically for one-step majority voting in decoding step. We perform actual addition instead of using the XOR operation to generate the resultant u vector. It is worth noting that the multiplications will still be performed using AND operators, as we need a binary result after the multiplication operation. Moreover, if we perform a non-binary multiplication instead, we will be required to perform a modular reduction by 2 to convert each element into binary after multiplication.

Note: Matrices are accessed in row-major or column-major order and vectors are treated as row or column depending on the order of the matrix and the dimension of the vector.

2) ***Vector Addition****:* Vector addition, as shown in Figure 2, is required only while performing the encryption operation. Its function is to add a random error vector to the encoded message for further obfuscation. The codeword and error vector are both binary and we further lower the hardware cost by performing additions without using adders. We leverage XOR to implement the required bit-wise vector addition operation.

3) ***Matrix Multiplication****:* The key generation module will need to perform two matrix multiplication operations. We extend the matrix-vector multiplication (binary version) module to perform matrix-matrix multiplication instead. As shown in Figure 2, the encoding matrix G will be accessed row-wise and the permutation matrix P will be accessed column-wise. Unlike in matrix-vector multiplication, where a vector loaded once for multiplication will be retained throughout, in matrix multiplication we will need to load rows as many times as number of columns instead.

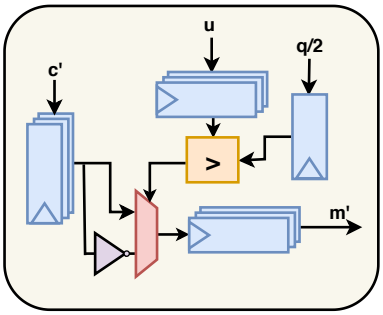

Fig. 3. Error correction using one-step majority voting.

4) ***Error Correction****:* We leverage the conditional operator, mux to perform error correction through one-step majority voting. The hardware circuit is shown in Figure 3. Error correction is performed by comparing the non-binary elements in vector u to a scalar with value $\lfloor \frac{q}{2} \rfloor$. Here, q is the prime number chosen while setting up the OLS codes for the cryptosystem. If an element in the i^{th} position in u is greater than $\lfloor \frac{q}{2} \rfloor$, then the corresponding i^{th} element in the codeword is erroneous. Error-correction is performed on this element by flipping it. Likewise, if the i^{th} element in u is less than $\lfloor \frac{q}{2} \rfloor$ then the corresponding i^{th} element in the codeword is correct and needs to be accepted as is. Error-correction performed this way is known as error correction through one-step majority voting.

V. Complexity Assessment

We present a brief evaluation of the proposed cryptosystem's complexity based on a similar approach as proposed by Baldi et al.

[18]. Multiplication of the scrambling matrix, S, the generator matrix G, and the permutation matrix, P contributes to the key generation complexity. KeyGen's complexity can be expressed as:

$$C_{KeyGen} = C_{mul}(SG) + C_{mul}(G''P) \tag{4}$$

Here, $C_{mul}(SG)$ represents the number of operations needed for computing the product of S and G, and G'' denotes the result after multiplying S and G. Encryption complexity is due to the multiplication of the message by the public key, G', and then addition of a k-bit error vector to the computed product. This addition operation will incur k binary operations. Encryption complexity, C_{Enc}, can be expressed as:

$$C_{Enc} = C_{mul}(mG') + k \tag{5}$$

The decryption complexity needs to be divided into three parts, which can be expressed as follows:

$$C_{Dec} = C_{mul}(cP^{-1}) + C_{mv} + C_{mul}(m'S^{-1}) \tag{6}$$

Here, $C_{mul}(cP^{-1})$ is the number of operations required to compute cP^{-1}. $C_{mul}(m'S^{-1})$ is the number of operations required to compute $m'S^{-1}$, and C_{mv} is the number of operations required for decoding through majority voting, which can be further expressed as:

$$C_{mv} = C_{mul}(c'H) + C_{mul}(uH) + n \tag{7}$$

Here, $C_{mul}(c'H)$ is the number of operations required to compute $c'H$. $C_{mul}(uH)$ is the number of operations required to compute uH, where u denotes the result of multiplying $c'H$ and n binary operations are required to perform a check on the bits for error and perform error-correction if required.

VI. Performance Evaluation

For evaluating the performance of the proposed McEliece cryptosystem, we implemented our design in Verilog and carried out synthesis using Xilinx Vivado 2018.2 design suite on a Xilinx CLG400ACX1341 Zynq-7000 FPGA board.

TABLE II
Correlation between k & n and Latency & Area for KeyGen, Enc and Dec Modules

Operation	Hardware Cost(LUTs)	Latency
KeyGen	$k+n$	$2kn$
Enc	$k+n$	$2n$
Dec	$k+2n$	$k+3n$

In Table II, we present an empirical analysis of the chosen parameters $\{k, n\}$ and the hardware cost and latency for all three modules of the McEliece cryptosystem co-processor. Both hardware cost and latency can be computed as functions of the code parameters k and n. Observing the hardware cost and latency trends not only provide key insights but also help strategizing the new design implementations.

Table III presents the hardware cost and latency, in clock cycles, for the entire co-processor with varying k and n values. The choice of these parameters will define the size of registers, muxes, gates, and datapath in the co-processor. As k and n values increase, LUT and register utilization increase accordingly. It is worth noting that our implementation has zero DSP cost, because we did not perform any actual multiplication operations throughout our implementation. The implementation is not optimized using BRAMs, we plan to do it as future work. Latency is also dependent on k and n, as the number of operations to be performed is defined by the size of the encoding and decoding matrices which are of the order k and n. Figure 4 and 5 show the increasing hardware cost and latency trend for our co-processor with various error correction t capabilities.

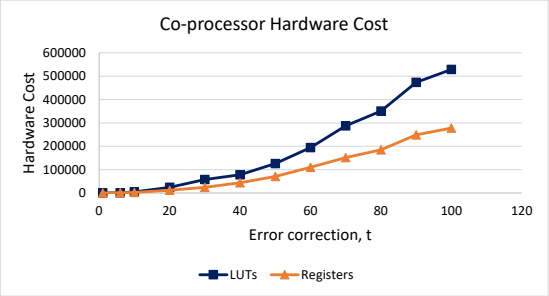

Fig. 4. Hardware Cost for the Co-processor.

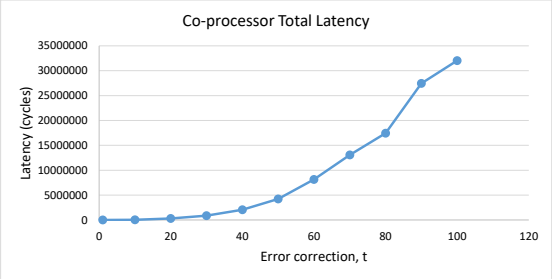

Fig. 5. Total Latency (cycles) for the Co-processor.

TABLE III
Hardware Cost and Latency(cycles) for the Proposed Co-processor with Different k and n

Parameters		Hardware Cost		Latency		
k	n	LUTs	Registers	KeyGen	Enc	Dec
361	741	4,764	2,626	28,158	1,482	2,584
1681	3321	24,174	11,745	272,322	6,642	11,644
3481	7021	57,492	24,646	828,478	14,042	24,544
6241	12561	77,981	44,017	1,984,638	25,122	43,924
10201	20301	125,787	71,182	4,100,802	40,602	71,104
16129	31369	194,470	110,356	7,967,726	62,738	110,236
22201	43061	287,752	151,459	12,832,178	86,122	151,384
26569	52649	350,981	184,609	17,163,574	105,298	184,516
36481	70861	473,240	249,185	27,068,902	141,722	249,064
39601	79401	528,567	277,921	31,601,598	158,802	277,804

Next, we compare the key length of our proposed OLSC-based scheme to a conventional binary Goppa code-based scheme. The claim made in Section III about a shorter key length using orthogonal Latin square codes is further confirmed by Figure 6. Finally, we compare the performance of our implementation with some of the existing hardware implementations of the classic McEliece cryptosystem. The purpose of this comparison is to highlight the advantages of replacing the binary Goppa code with the OLS codes. Tables IV, V, and VI summarize our comparisons with related work. Most of the implementations provide 80-bit security; only one implementation

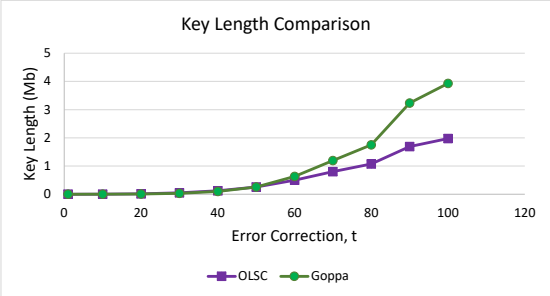

Fig. 6. Comparison of Key Length for the Co-processor.

provides 103-bit security. Hardware implementations with higher security levels for the classic McEliece cryptosystem are not available because of high complexity, large key sizes and limited memory availability.

TABLE IV
COMPARISON OF SECURITY, PARAMETERS AND PUBLIC-KEY SIZE

Work	Security	Parameters(n, k, t)	Public-key Size (Kb)	Configurable parameters
[13]	80-bit	2048, 1751, 27	3502	No
[14]	80-bit	2048, 1751, 27	3502	No
[15]	80-bit	2048, 1751, 27	520	Yes
[16]	103-bit	2048, 1498, 50	824	No
Ours	112-bit	20301, 10201, 50	2564	Yes

Table V and VI present the hardware cost and latency of [13], [14], [15], and [16]. We compare these with our architecture, generating results using similar parameters. We observe that our implementation provides higher security at a much lower hardware cost than the other implementations.

TABLE V
COMPARISON OF HARDWARE COST(LUTS)

Work	Platform	Hardware Cost			
		KeyGen	Enc	Dec	Total
[13]	Spartan-3	-	668	11,218	-
[14]	Spartan-3	-	-	2,979	-
[15]	Spartan-3	-	-	785	-
Ours	Spartan-3	12439	762	12786	25987
[16]	Virtex-5	-	-	-	14,537
Ours	Virtex-5	11900	584	12290	24774

TABLE VI
COMPARISON OF LATENCY(IN CLOCK CYCLES)

Work	Latency		
	KeyGen	Enc	Dec
[13]	-	7,889,200	891,736
[14]	-	-	94,249
[15]	-	-	87,363
[16]	14,670,000	81,500	210,280
Ours	4,100,802	40,602	71,104

Latency depends completely on the number and complexity of the operations being performed. Our decryption operation is about $12\times$, $1.5\times$ and $1.2\times$ faster than the decryption in [13], [14], and [15] respectively. Moreover, our key generation, encryption, and decryption operation is about $3.7\times$, $2\times$, and $3\times$ faster respectively than the respective operations in [16]. The faster decryption module is the result of the one-step majority decoding technique of OLS code.

VII. SECURITY ANALYSIS

The security of the proposed variant of the McEliece cryptosystem co-processor depends on how difficult it will be for an adversary, who knows the public key G' and can intercept the ciphertext, to determine the plaintext message m. To determine m, an adversary can try two different types of attacks. First, he/she can try to recover the generator matrix G from G' using the key recovery brute-force attack. Second, he/she can perform a message recovery attack in an attempt to recover m from the intercepted ciphertext without learning anything about G. Additionally, an attacker can perform an attack by trying to find a codeword with minimum hamming distance from all the possible codewords.

Key Recovery Attack: A Key recovery attack or a structural attack is the first category of attack that may seem promising as an adversary can try the brute-force method to recover G. Hence, if we can prove that the key space is large enough for the chosen parameter values so that it is practically impossible to recover G, then we can justify the security of the proposed scheme. Therefore, in this section, we will explore the key space with an illustrative example code parameter set. Hsiao et al. proved an important theorem in their work [17] on OLSC. According to this theorem, for an existing set of λ_q $q\times$ q orthogonal Latin squares ($\lambda_q \leq$ q - 1), there exists a t-error correcting code, where

$$t = \lfloor \frac{\lambda_q}{2} \rfloor + 1 \tag{8}$$

Equation 8 can also be interpreted in a way to predict the number of orthogonal Latin squares that will be used in the encoding matrix by rewriting it as follows:

$$\lambda_q = 2t - 2 \tag{9}$$

Knowing equation 9, we can discuss the illustrative example with q = 11. λ_q will be 10 with q = 11 and maximum bits that can be error corrected is t = 6. Table VII shows the number of ways orthogonal Latin squares can be picked to form the encoding matrix G. It is evident from the table that for even a small value of q, G can be formed in approximately 3 million different ways. It is worth noting that we have completely ignored the different possibilities that exist for generating the non-singular matrix S as well as permutation matrix P for the given parameters.

TABLE VII
KEY SPACE EXPLORATION ILLUSTRATION

No. of bits to be error-corrected, t	No. of Orthogonal Latin Squares, $2t-2$	No. of ways to pick orthogonal Latin squares
1	0	0
2	2	90
3	4	5040
4	6	151,200
5	8	1,814,400
6	10	3,628,800

In a practical scenario, the selected error-correction code will be required to correct up to 100 bits of errors. A suitable q is 199, as we will need at least 198 orthogonal Latin squares to perform error correction on 100-bits. And, if we use all 198 orthogonal Latin squares to compute the generator matrix, the key space is as large as 198!, again ignoring the possibilities for the S and P matrices. Hence, if q and t are large enough, there are so many possible ways of constructing G that there is no chance of a one-to-one mapping from G to G'. Therefore, the idea of recovering G by brute-forcing seems impossible to an adversary. Hence, a structural attack that recreates a private key against a cryptosystem based on the orthogonal Latin squares codes is infeasible.

Message Recovery Attack: As learning the secret key G from the public key is impractical, an attempt to recover m from the ciphertext without learning G may prove a more promising approach for an adversary. But deciphering m without G requires solving the basic problem of decoding an arbitrary (n, k) linear code in the presence of t errors. As discussed earlier in Section I, the decoding problem for linear codes is, in general, NP-hard [19] and so if the code parameters are large enough, then this attack, too, will be infeasible.

Codeword Finding Attack: The idea behind this attack is to compare the hamming distance of the received word with all possible codewords for the given code. The attacker starts by enumerating through all possible codewords and simultaneously computes the hamming distance, d_H, between the received word and the codeword. The attacker returns the codeword with the minimum d_H as success. The attacker, while trying to compare the received word with each possible codeword in the code, requires 2^k attempts as there will be 2^k possible codewords. Hence, the complexity of this attack will be $O(n2^k)$. Considering $k = 361$, an adversary would require at least $2^{361} \approx 10^{109}$ attempts. Typical values for k are much larger in practice, making the attack impossible.

Based on this discussion, we can conclude that any brute-force approach against the proposed variant of McEliece cryptosystem is too complex to be successful. We would also like to highlight here that we choose q as a prime number to avoid attacks exploiting non-prime numbers.

VIII. Conclusion

In this paper, we presented the design of a new variant of the McEliece cryptosystem using OLS codes. The proposed scheme is much faster than an equivalent binary Goppa code-based system due to one-step decoding. Moreover, an efficient and lightweight FPGA-based implementation of the proposed co-processor was also presented. Design choices in the implementation of the sub-modules within the cryptosystem's co-processor led to optimal hardware cost and latency for a wide range of parameter set. When compared to other existing classic McEliece cryptosystems, our implementation was found to be $3.3\times$ faster on an average. As a future work, we would like to further optimize our current implementation. We would also like to perform a formal verification of the security provided by the OLSC-based variant to confirm that the proposed McEliece cryptosystem can serve as an efficient and high speed variant of the original McEliece cryptosystem.

References

[1] W. Knight, "Ibm raises the bar with a 50-qubit quantum computer," *Sighted at MIT Review Technology: technologyreview. com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer*, 2017.

[2] J. Hsu, "Ces 2018: Intel's 49-qubit chip shoots for quantum supremacy," *IEEE Spectrum Tech Talk*, 2018.

[3] R. Courtland, "Google aims for quantum computing supremacy [news]," *IEEE Spectrum*, vol. 54, no. 6, pp. 9–10, 2017.

[4] S. Daily. (2018) World-first quantum computer simulation of chemical bonds using trapped ions. [Online]. Available: sciencedaily.com/releases/2018/07/180724110028.htm

[5] D. Aharonov, M. Ben-Or, E. Eban, and U. Mahadev, "Interactive proofs for quantum computations," *arXiv preprint arXiv:1704.04487*, 2017.

[6] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[7] R. Agrawal, L. Bu, A. Ehret, and M. A. Kinsy, "Open-source fpga implementation of post-quantum cryptographic hardware primitives," in *Field Programmable Logic and Applications (FPL), 2019 International Conference on*, Sep. 2019.

[8] R. Agrawal, L. Bu, and M. A. Kinsy, "A post-quantum secure discrete gaussian noise sampler," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020.

[9] ——, "Fast arithmetic hardware library for rlwe-based homomorphic encryption," *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2020.

[10] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 1996, pp. 212–219.

[11] NIST. (2019) Pqc standardization process. [Online]. Available: csrc.nist.gov/news/2019/pqc-standardization-process-2nd-round-candidates

[12] R. J. Mceliece, "A public-key cryptosystem based on algebraic," *Coding Thv*, vol. 4244, pp. 114–116, 1978.

[13] T. Eisenbarth, T. Güneysu, S. Heyse, and C. Paar, "Microeliece: Mceliece for embedded devices," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2009, pp. 49–64.

[14] S. Ghosh, J. Delvaux, L. Uhsadel, and I. Verbauwhede, "A speed area optimized embedded co-processor for mceliece cryptosystem," in *2012 IEEE 23rd International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 2012, pp. 102–108.

[15] P. M. C. Massolino, P. S. Barreto, and W. V. Ruggiero, "Optimized and scalable co-processor for mceliece with binary goppa codes," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 3, p. 45, 2015.

[16] A. Shoufan, T. Wink, G. Molter, S. Huss, and F. Strentzke, "A novel processor architecture for mceliece cryptosystem and fpga platforms," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2009, pp. 98–105.

[17] M. Hsiao, D. Bossen, and R. Chien, "Orthogonal latin square codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 390–394, 1970.

[18] M. Baldi, M. Bodrato, and F. Chiaraluce, "A new analysis of the mceliece cryptosystem based on qc-ldpc codes," in *International Conference on Security and Cryptography for Networks*. Springer, 2008, pp. 246–262.

[19] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.