Hardware-Software Implementation of a McEliece Cryptosystem for Post-Quantum Cryptography

Mariano López-García[1], Enrique Cantó-Navarro[2]

[1] Universitat Politècnica de Catalunya, 08800 Vilanova I la Geltrú, Spain
[2] Universidad Rovira i Virgili, 43007 Tarragona, Spain
mariano.lopez@upc.edu

Abstract. This paper shows the implementation on FPGA of a McEliece cryptosystem, which ensures the security recommendations given by the European Telecommunications Standards Institute (ETSI) for next generation of quantum-resistant cryptosystems. The proposed implementation provides more than 128 bits of quantum security using a public key of 2,097,152 bytes. The proposed system is based on a hardware/software implementation that uses an ARM Cortex-A53 core connected to a coprocessor through an AX14 lite interface. The complete system was implemented on a Xilinx Zynq UltraScale+ and it is able to decipher texts of 8192 bit-length is 47.39 ms.

Keywords: McEliece, post-Quantum crytography, FPGA, hardware/software co-design, embedded devices.

1 Introduction

Robert J. McEliece proposed in 1978 a code-based public-key cryptosystem whose security is proven to be NP-complete [1][2]. Even its promising properties, in the field of embedded systems its use was very reduced, due to the memory requirements needed to store both private and public keys. This was, initially, a clear disadvantage against other cryptosystems based, for instance, on RSA or elliptic curves. Nowadays, with the advance of the microelectronic industry, most commercial brands provide embedded platforms that already include large memories that allow its implementation with no restrictions.

McEliece cryptosystem can achieve the same security level against attacks performed on classic computers than those based on RSA or Elliptic curves, but however, it is able to perform encryption and decryption 25 and 2 times faster than RSA, respectively [3]. In fact, it is considered a secure cryptosystem against quantum attacks. Indeed, the National Institute of Standards and Technology (NIST) included McEliece among the public-key cryptosystems listed in the document "Report on Post-Quantum Cryptography" [4]. On the other hand, the European Telecommunications Standards Institute (ETSI) issued a report that not only includes McEliece in the list of quantum save security cryptosystems, but also recommended for this algorithm the use of public keys of 1,046,739 bytes to obtain 131-bits of quantum security level [3]. Similarly, the European project "PQCrypto: Post-quantum cryptography for long-term

security" also recommends, for public key encryption, the use of McEliece as quantum-resistant cryptosystem [5].

The original proposal of McEliece was based on binary Goppa codes, although this scheme can be extended to any family of linear codes. In this way, in order to decrease the size of the keys, several proposals were presented in the past. However, until now there is no clear evidence that using others codes it was possible to increase the security of this scheme.

Obtaining the public key of a McEliece cryptosystem consists in choosing two parameters: n and t. These two integers ($n=2^m$ is much bigger than t) are used to create the generator matrix G of dimension k x n of a binary irreducible Goppa code ($k=n-m \cdot t$). The parameter t is the number of errors that can be corrected by the code, n is the length of the cypher text, and k is the size of the block to be encrypted (message). Matrix G is hidden by using a permutation P matrix of dimension n x n and a substitution matrix S of dimension k x k. Both matrices are chosen at random in such a way that the public key G^{pub} is obtained as $G^{pub}=S \cdot G \cdot P$. On the other hand, G, S and P matrices, forms the private key.

One of the weaknesses of the original McEliece cryptosystem is that it fails when encrypting messages that have some known relationship between them [6]. However, such a problem can be fixed by scrambling or randomizing the messages in order to eliminate their dependency [7]. Though it is not easy to choose the best parameters for n and t, Bernstein [8] *et al*. proposed concrete parameters for a variant of McEliece cryptosystem called *CCA2-secure*, that was attacked applying the list decoding algorithm and using a classic computer based on a 2.4GHz Core 2 Quad CPU. From the obtained results, authors conclude that for a non-quantum security level of 256 bits Goppa codes of length n=6624, t=115 and k=5129 should be used, which leads to a public key size of 958,481 bytes (for the variant of McEliece CCA2-secure, the public key could be reduced to $k \cdot (n-k)$ bits instead of $n \cdot k$ bits).

In the past, there have been several proposals for the implementation of McEliece in hardware. In [9] this cryptosystem was implemented on both an 8-bit AVR microcontroller and a Xilinx Spartan-3 FPGA. In this proposal, the matrix S is not stored and it is created using a PRNG (Pseudorandom Number Generator). The implementation achieved a maximum throughput for encryption of 1,626,517 bits/s (calculated by extrapolation on a DDR2 memory) and 3,899 bits/s for the FPGA and 8-bit AVR implementations, respectively. In the implementation proposed by Bernstein in [10], performed on an Intel Core 2, the best rate of cycles per byte obtained was 2260 for decryption with n=2048 and t=40. Other authors as Von Maurich and Güneysu [11], proposed a lightweight implementation of McEliece by using quasi-cyclic moderate density parity-check codes. Their system was implemented on a Virtex 6 FPGA using a non-quantum security level of 80 bits. Such system was able to encrypt and decrypt an input block in 2.2ms and 13.4 ms, respectively. In [12] Ghosh *et al*. presented a hardware-software (Hw/Sw) co-design implemented on a Spartan 3 FPGA that optimizes speed and area. Authors claimed that their solution is faster and smaller (in terms of area) that existing designs, taking less than 100k clock cycles for decryption. In fact, it is difficult to perform a fair comparison between all these publications, since the goals to be optimized are very different as well as the performance of the embedded platforms or the security level (quantum or non-quantum) used in their implementations.

This paper presents the implementation on an embedded device of a complete McEliece cryptosystem on dedicated hardware. The security level is in accordance with the recommendations given by ETSI for post-quantum resistant cryptosystems. The implementation, which uses as parameters $n=2^m=8192$, $m=13$, $t=315$ and $k=m-n\cdot t=4097$, is based on a hardware-software co-design that includes an ARM Cortex-A53 microprocessor and a reconfigurable hardware. The overall system is integrated on a Xilinx Zynq UltraScale+ FPGA. The encryption and decryption processes are carried out in 1.55 ms (throughput of 5.28 Mbits/s) and 47.39 ms (throughput of 86.45 kbits/s), respectively.

This paper is organized as follows. Section 2 describes the basic theory about the McEliece Cryptosystem. Section 3 presents the hardware-software partitioning that was performed. Section 4 describes the internal structure of the McElice IP block implemented in hardware. Section 5 shows the experimental results and finally conclusions are presented.

2 McEliece Cryptosystem Review

The fundamentals of McEliece Cryptosystem are well documented in [1][13][15], so that they will be only briefly reviewed here for understanding the internal structure of the implemented coprocessor and how the software-hardware partitioning is performed. As in the original proposal, the proposed implementation of McEliece is based on a binary irreducible Goppa code. Thus, operations (sums and products) performed with binary matrices are computed over $\mathbb{F}_2$, while the coefficients of a Goppa polynomial are defined in $\mathbb{F}_{2^m}$.

2.1 Key Generation

The key generation consists in choosing the values of n and t; in our case, this choice is carried out according to the compliance of safety recommendations given by ETSI for post-quantum cryptosystems. As mentioned before, such values are $n=2^m=8192$, and $t=315$. Then, the following matrices are generated:

- G: a k x n generator matrix of a binary irreducible Goppa code C that has minimum distance d≥2t+1, being $k=n-m\cdot t=4097$.
- S: a k x k binary non-singular matrix over $\mathbb{F}_2$ chosen at random.
- P: a n x n permutation matrix over $\mathbb{F}_2$ chosen at random with exactly one 1 in every row and column, and the rest of entries 0.
- G^{pub}: A k x n matrix obtained by the product $S\cdot G\cdot P$.

The public key is then defined by the pair (G^{pub}, t). It is important to remark that the permutation matrix P is chosen to create a generator matrix G of the kind $G = (J^T \mid I_{n-k})$, where I_{n-k} is the identity matrix. See [14] for a wider explanation.

2.2 Encryption Process

The message m to be encrypted is k bits length as it is created as follows:

- Chose a random $z \in \mathbb{F}^n$ with the property that z has t entries different from zero, and n-t entries equal to zero.
- Compute the encrypted message r as $r = m\ G^{pub} + z$, in which the vector z is interpreted as the error.

2.3 Decryption Process

The cipher-text decryption is performed using the private key (S,D,P), in which S and P are matrices found during the key generation; and D is an algorithm for decoding the irreducible Goppa code C. The original message is obtained following the steps:

- Compute the string $y = r \cdot P^{-1} = (m\ G^{pub} + \varepsilon)\ P^{-1} = m\ S\ G + \varepsilon\ P^{-1}$.
- As y is a codeword of C, use the decoding algorithm D to find y' as $D(y) = D(mSG + \varepsilon P^{-1} = mSG = y'$. This decoding is performed using the Patterson algorithm.
- It is observed that $y' = m(SG) = (mS)G = m'G$, where $m' = mS$ is another message of the same length. Therefore, y is the codeword associated to message m'.
- As P was chosen to create a matrix G of the form $G = (J^T \mid I_{n-k})$, therefore the last n-k coordinates of y' are the message m' (truncation).
- Once we have m' we multiply by S^{-1} to obtain $m = m'S^{-1} = mSS^{-1}$.

Patterson Algorithm

The Patterson algorithm is used to computer the error-locator polynomial $\sigma(Z)$ of a codeword with errors. The polynomial $\sigma(Z)$ of a vector y satisfies the property that $\sigma(\alpha_i)=0$ if and only if y has an error in the i-th position. The algorithm is based on the knowledge of the syndrome $S_y(Z)$ of the codeword, and it is computed following the steps:

1. Compute the syndrome $S_y(Z)$.
2. Compute the inverse $T(Z) = (S_y(Z))^{-1}$.
3. Compute the square root $R(z) = sqrt(T(Z)+Z)$.
4. Solve the equation $a(Z) = b(Z) \cdot R(Z)$.
5. Compute the error location polynomial $\sigma(Z) = a^2(Z) + Z \cdot b^2(Z)$.
6. Compute the error vector ε from $\sigma(Z)$.
7. Correct the codeword $y' = y + \varepsilon$.

The inverse $T(Z)$ and the resolution of equation $a(Z) = b(Z) \cdot R(Z)$ are computed using the Extended Euclidean Algorithm.

Extended Euclidean Algorithm

The Extended Euclidean algorithm computes the greatest common divisor of two polynomials $a(Z)$ and $b(Z)$, along with two additional polynomials $\lambda(Z)$ and $\mu(Z)$ satisfying:

$$\lambda(Z)a(Z) + \mu(Z)b(Z) = gcd(a(Z), b(Z)) \quad (1)$$

The two initial polynomials have *deg(b)≥deg(a)*. The algorithm works as follows. First, *b(Z)* is divided by *a(Z)* obtaining a quotient *q(Z)* and a reminder *r(Z)* that meet *b(Z)=a(Z)a(Z)+r(Z)* and *deg(r)<deg(a)*. Since *gcd(a(Z),b(Z))=gcd(r(Z),a(Z))*, we can reduce the problem of finding *gcd(a(Z),b(Z))* to determining *gcd(r(Z),a(Z))* where the degree of *r(Z)* is smaller than the degree of *b(Z)*. The process could be repeated until one of the arguments is zero.

Extended Euclidean Algorithm

Input: $(r_0, r_1, \lambda_0, \lambda_1, \mu_0, \mu_1):=(a, b, 1, 0, 0, 1)$
Output: (g, λ, μ)

while $r_i \neq 0$ ***do***
    q_i*=: quotient or* r_{i-1} *on division by* r_i*;*
    $(r_{i+1}, \lambda_{i+1}, \mu_{i+1}) := (r_{i+1}, \lambda_{i+1}, \mu_{i+1}) - q_i \cdot (r_{i+1}, \lambda_{i+1}, \mu_{i+1})$*;*
    i:=i+1;
end while*;*
return $(g, \lambda, \mu):=(r_{i+1}, \lambda_{i+1}, \mu_{i+1})$*;*

3 Hardware-Software Partitioning

The architecture of a system based on hardware-software co-design includes a microprocessor, which purpose is managing the organized execution of the main program, the communication between input/output devices and the control of information through the system's buses. The specific subsystems designed on reconfigurable hardware, known as coprocessors, are connected to the microprocessor by means of the bus architecture. Due to their specific design, coprocessors have the ability of resolving some specific functionalities, regarding some part of the application or algorithm, in an efficient way and in a very short time. However, they are distinguished with a low flexibility, in the sense that only are able of resolving the task for what they have been designed. The concept of partitioning consists in determining which parts of an algorithm are best suited for execution by software (microprocessor) or for synthesis in dedicated hardware (coprocessors), in order to meet a set of constraints and goals. Usually, the parameter to be optimized is the resolution time while the main constraint is usually the area needed by the coprocessor. Additionally, the success of the hardware-software co-design depends not only on the active cooperation between the software and hardware modules, but also on the communication protocol needed for their proper operation.

Table 1 shows the execution time when encrypting and decrypting a block of 512 characters in length chosen at random. The table also includes the specific time needed by each of the steps described in section 2. These results were obtained programming the overall McEliece cryptosystem in C language.

	Function	*No. Cycles*	*Time (ms)*
Encryption	Compute m G^{pub}	$166.53 \cdot 10^4 \cdot T_{CLK}$	1.51 ms
	Add error: r=m G^{pub}+z	$497.42 \cdot 10^2 \cdot T_{CLK}$	0.045 ms
Total time for encryption:		$166{,}65 \cdot 10^4 \cdot T_{CLK}$	1.515 ms
Decryption	Compute $r \cdot P^{-1}$	$186.95 \cdot 10^3 \cdot T_{CLK}$	0.169 ms
	Syndrome Polynomial $S_y(Z)$	$350.85 \cdot 10^4 \cdot T_{CLK}$	3.189 ms
	Inverse Syndrome $T(Z)= (S_y(Z))^{-1}$	**$347.94 \cdot 10^6 \cdot T_{CLK}$**	**316.31 ms**
	Square root R(z)=sqrt(T(Z)+Z)	$124.14 \cdot 10^5 \cdot T_{CLK}$	11.28 ms
	Equation $a(Z)=b(Z) \cdot R(Z)$	**$184.45 \cdot 10^6 \cdot T_{CLK}$**	**167.68 ms**
	Compute $\sigma(Z)=a^2(Z)+Z \cdot b^2(Z)$	$112{,}56 \cdot 10^5 \cdot T_{CLK}$	10.23 ms
	Compute vector error ε	**$537.72 \cdot 10^6 \cdot T_{CLK}$**	**488.84 ms**
	Correct the codeword y'=y+ ε	$189.02 \cdot 10^3 \cdot T_{CLK}$	0.171 ms
	Truncation y' to obtain m'	$873.31 \cdot 10^2 \cdot T_{CLK}$	0.079 ms
	Compute $m=m' \cdot S^{-1}$	$461.34 \cdot 10^4 \cdot T_{CLK}$	4.19 ms
Total time for decryption:		$110.23 \cdot 10^7 \cdot T_{CLK}$	1002.1 ms

Table. 1. Execution time for each step of the McEliece Cryptosystem (encryption and decryption). Results are given in clock cycles and milliseconds using an ARM Cortex A53 microprocessor clocked at 1,1 GHz. In bold are represented the steps that will be implemented in hardware.

As can be seen, the calculation of the inverse syndrome, the polynomial *a(Z)* and the vector error ε takes about the 98% (972.8 ms or 1070 $\cdot 10^6$ clock cycles) of the total time. Thus, these three steps are the candidates to be implemented in hardware.

On the other hand, the partitioning could be performed at different levels, ranging from simple operations or instructions (fine granularity) to complex processes or routines (coarse granularity). After analyzing the most time consuming steps, it seems that the optimal partitioning should be performed mixing different granularities. Thus, a unique coprocessor is designed, which internally includes several subsystems for implementing some basic operations. The processor is able to perform operations using Goppa polynomials. The proper sequence in which such operations should be performed depends on the step that is being implemented by the coprocessor. Such a sequence is generated by managing several internal signals by a control unit, which conveniently programmed stablishes the order in which the subsystems are activated. The internal subsystems are also designed in a similar way including each of them its corresponding control unit.

4 Coprocessor Design (McElice IP)

Fig 1. shows the internal architecture of the McEliece IP block. Such IP is connected as slave peripheral to the AXI4 bus, which is used during the configuration process performed by the microprocessor. Additionally, the IP is also attached to an AXI4 stream-interface used for retrieving and storing polynomial data from/to memory. In order to improve the performance, a couple of buffers are included. Such buffers store

(temporally) the input/output Goppa polynomials (input operands) while the arithmetic unit is busy processing data.

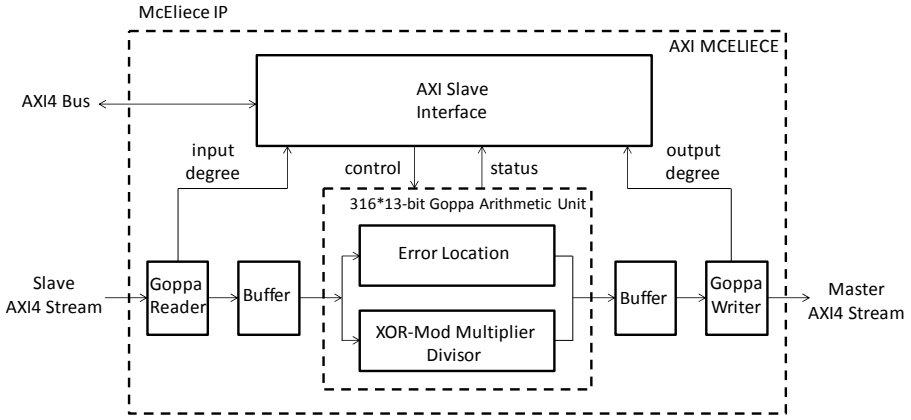

Fig. 1. Internal architecture of McEliece IP.

The coprocessor is designed to carry out seven different operations using Goppa polynomials (see table 2). Such operations are indicated by writing a specific code in the input registers, which are used to communicate the microprocessor and the IP block.

Codification	Operation
0000	$op_1 * op_2$
0001	$(op_1 * op_2) \bmod G_p$
0010	$(op_1 * op_2) \oplus op_3$
0011	$((op_1 * op_2) \oplus op_3) \bmod G_p$
0100	op_1 / op_2
1000	Error Location(op_1)
1001	Not used
1110	Set G_p

Table. 2. Codification table and Goppa polynomial operations performed by the arithmetic unit. Note: G_p is the Goppa polynomial generator used to create the private and public keys.

All these operations can be performed designing only four basic operations: multiplication, division, addition (xor) and modulus.

The Goppa multiplier is presented in Fig. 2. Note as its design includes 316 (t=315) internal multipliers in $\mathbb{F}_2$ that are able to perform in parallel the 316 multiplications between the coefficients that form the two input polynomials. An additional block is added to perform the *xor* operation between this multiplication and a third polynomial when necessary. This additional operation is very common in the Extended Euclides Algorithm.

As shown in Fig. 3.a, a Goppa divider can be designed by including a Goppa multiplier and a divider in $\mathbb{F}_2$. Finally, the error location module can be implemented by simply using a multiplier in $\mathbb{F}_2$ (see Fig. 3.b).

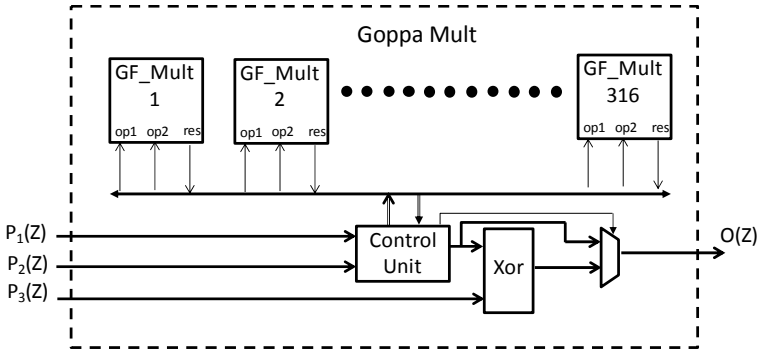

Fig. 2. Internal architecture of Goppa Multiplier.

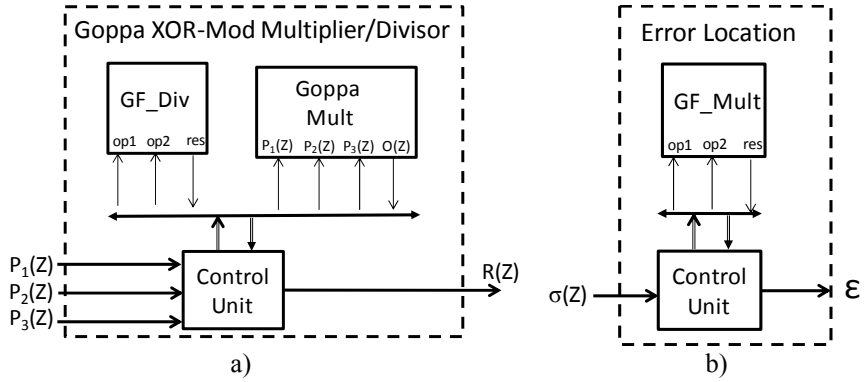

Fig. 3. Internal architecture of: a) Goppa Xor-Mod Multiplier-Divisor, b) Error Location.

5 Experimental Results

The overall embedded system was implemented on a Xilinx Zynq UltraScale+ ZCU 102 evaluation board, which includes an MPSoC device XCZU9EG-2FFVB11561. Although the MPSoC includes four ARM Cortex-A53 core clocked at 1.1 GHz only one of them is used in the final implementation. Additionally, a 4GB DDR4 64-bit memory is available, so that there is no limitations on the size of the keys to be stored. All the hardware designs were described in VHDL and implemented by using the software tools provided by Xilinx. The results are shown in Table 3.

Table 4 shows the results for the execution time of the overall hardware-software co-design proposed in this paper. Results are obtained when encrypting a block of 4097 bits (512 characters). Afterward the cypher text of 8192 bits (1024 characters) is decrypted recovering the original message. The table also shows the acceleration factor, provided by the McEliece IP block, for those steps implemented in hardware against the software version. In order to meet the maximum frequency requirements imposed by the critical path, the McEliece IP block is clocked at 250 MHz. Note that the inverse

of the syndrome is processed in 15.67 ms, which means that it is about 20.18 faster than the software execution. As similar figure is obtained when calculating the polynomial *a(Z)*. This polynomial is computed by the coprocessor in 7.38 ms, leading to an acceleration factor of 22.72. The vector error calculation offers the best result. This vector is obtained in 5.21 ms against the 488.84 ms needed by the microprocessor (acceleration factor of 93.82). Thus, a complete decryption could be performed in approximately 48 ms.

	Used	Available	Utilization %
LUTs	77357	274080	28.22
FFs	41338	548160	7.54

Table. 3. Resources used by the McEliece IP Block when implemented on Zynq UltraScale+™ MPSoC family.

Function	Time (ms) (Only Software)	Time (ms) (Hardware-Software)	Ratio: Acceleration factor
Compute m G^{pub}	1.51 ms	1.51 ms (sw)	--
Add error: r=m G^{pub}+z	0.045 ms	0.045 ms (sw)	--
Compute $r \cdot P^{-1}$	0.169 ms	0.169 ms (sw)	--
Syndrome Polynomial $S_y(Z)$	3.189 ms	3.189 ms (sw)	--
Inverse Syndrome $T(Z)= (S_y(Z))^{-1}$	**316.31 ms**	**15.67 ms (hw)**	**20.18**
Square root R(z)=sqrt(T(Z)+Z)	11.28 ms	11.28 ms (sw)	--
Equation $a(Z)=b(Z) \cdot R(Z)$	**167.68 ms**	**7.38 ms (hw)**	**22.72**
Compute $\sigma(Z)=a^2(Z)+Z \cdot b^2(Z)$	10.23 ms	10.23 ms (sw)	--
Compute vector error ε	**488.84 ms**	**5.21 ms (hw)**	**93.82**
Correct the codeword y'=y+ ε	0.171 ms	0.171 ms (sw)	--
Truncation y' to obtain m'	0.079 ms	0.079 ms (sw)	--
Compute $m=m' \cdot S^{-1}$	4.19 ms	4.19 ms (sw)	--
Total time for encryption:	**1.515 ms**	**1.515 ms**	
Total time for decryption:	**1002.1 ms**	**47.39 ms**	**21.14**

Table. 4. Comparison execution time for both: only software implementation (ARM Cortex clocked at 1.1GHz) and hardware-software co-design including McEliece IP (coprocessor) clocked at 250 MHz.

Table 5 shows the throughput achieved by the software-hardware co-design when encrypting a plain text and decrypting its corresponding cypher text. Due to the simplicity of the operations involved in the process, the best result is obtained when encrypting the original message 5.285 Mb/s.

It is difficult to perform a fair comparison against other publications carried out in the past, because each of them uses a different FPGA (with different resources), a

specific-limited memory RAM, higher or lower values are selected for *n*, *k*, *t* (and therefore bigger or slower levels of security) and a different number of coprocessors. Furthermore, the goal of some implementations could be to obtain a lightweight version to be embedded in a hardware with limitations in area and speed. Table 6 shows the results obtained for other authors when implementing in hardware the three steps presented in section 3 (inverse Syndrome, equation a(Z) and error computation). In order to perform a fair comparison, such results are given in clock cycles (independent of the clock frequency) along with the values of *n*, *k*, *t* used in the implementation. Note that no extrapolations can be performed, since there is no proportionality between such values and the number of clock cycles. Furthermore, it important to remark that the security level given by most authors is referred to non-quantum cryptography and is based on the publication presented in [8].

	Throughput	
	Only Software	Hardware/Software
Plaint text: 4097 bits (encryption)	5.285 Mb/s	5.285 Mb/s
Ciphertext: 8192 bits (decryption)	4.09 kb/s	86.45 kb/s

Table. 5. Throughput for both software-only and hardware/software implementations.

	MicroEliece [13]	Hw/Sw co-design [12]	Our design
Frequency /	80MHz	92MHz	250MHz
Throughput for decryption	161,829 bit/s	1,716,66 bits/s*	86,452 bits/sec
Security, (n, k, t)	80 bits**, (2048,1751,27)	80 bits**, (2048,1751,27)	>131 bits***, (8192,4097,315)
Inverse Syndrome $T(Z)= (S_y(Z))^{-1}$	625 cycles	1821 cycles	$391 \cdot 10^3$ kcycles
Equation $a(Z)=b(Z) \cdot R(Z)$	312 cycles	960 cycles	$1845 \cdot 10^3$ kcycles
Compute vector error ε	312328 cycles	57356 cycles	$1302 \cdot 10^3$ kcycles

Table. 6 Comparison of the McEliece decryption Implementations. *Result estimated by us. ** Non-quantum security level obtained from [8]. ***As we are using a public key size that is twice the recommendation given by ETSI [3], we assume that quantum security level would be higher than 131 bits.

6 Conclusions

This paper presented the implementation of a McEliece cryptosystem on a Xilinx Zynq UltraScale+ FPGA. The structure of the system was based on a hardware-software co-design, which consists of an ARM Cortex microprocessor and a McEliece IP block that includes several hardware subsystems. The microprocessor acts as master of the system, while the IP block processes the most time consuming steps involved when decrypting a cypher text. The speed processing is increased in average by a factor

of 21when compared with an only-software resolution. The parameters for n, k and t were chosen in order to provide a security level in accordance with the recommendations given by ETSI for post-quantum cryptography. In order to improve area and speed, a faster version of McEliece cryptosystem, particularly suited for FPGA implementations, will be chosen in a future work.

Acknowledgments

This work was supported by the Ministerio de Economía y Competitividad in the framework of the Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, project TEC2015-68784-R.

References

1. McEliece. R. J..: A public key cryptosystem based on algebraic coding theory. DNS progress report 43.44, 1978.
2. Berlekamp, E. R., McEliece, R. J.: On the inherent intractability of certain coding problems. IEEE Trans. Information Theory, 24(3): 384-386 (1978)
3. ETSI – European Telecommunications Standards Institute: Quantum Safe Cryptography (QSC); Quantum-safe algorithmic framework. ETSI GR QSC 001 v1.1.1(2016).
4. National Institute of Standards and Technology. Report on Post-Quantum Cryptography. Internal report 8105. http://dx.doi.org/10.6028/NIST.IR.8105 (2016).
5. Augot, D. et al: Initial recommendations of long-term secure post-quantum systems. Horizon 2020 ICT-645622. Revision 1. (2015).
6. Berson, T.: Failure of the McEliece public-key cryptosystem under message-resend and related-message attack, pp. 213-220, Springer-Verlag (1997).
7. Engelbert, D. Overbeck, R. and Schmidt, A.: A summary of McEliece-type cryptosystems and their security. http://eprint.iacr.org/2006/162.ps
8. Bernstein, D. J., Lange, T., Peters, C.: Attacking and Defending the McEliece Cryptosystem. International Sorkshop on Post-Quantum Cryptography. pp 31-46 (2008).
9. Eisenbarth, T., Güneysu, T., Heyse, S., Paar, C.: MicroEliece: McEliece for embedded Devices, International Conference on Cryptographic Hardware and Embedded Systems - CHES (2009)
10. Bernstein, D. J., Buchmann, J., Dahmen, E.: Post-Quantum Cryptography. Springer-Verlag, (2009)
11. Von Maurich, I., Güneysu, T.: Lightweight Code-based Cryptography: QC_MDPC McEliece Encryption on Reconfigurable Devices. Design, Automation & Test in Europe Conference & Exhibition (DATE) (2014).
12. Ghosh, S., Delvaux, J., Uhsadel, L., Verbauwhede, I.: A Speed Area Optimized Embedded Co-Procesor for McEliece Cryptosistem. IEEE 23rd International Conference on Application-Specific Systems, Architectures and Processors (2012).
13. Heyse, S.: Code-based Cryptography: Implementing the McElice Scheme on Reconfigurable Hardware. Master Thesis, Faculty of Electrical Engineering and Information Technology, Ruhr-University Bochum (2009).
14. Flexiprovider. http://www.flexiprovider.de/
15. Quantum-resistant cryptography. Oriol Farràs. Technical Report. 2017.