# BLAKE-512 Based 128-bit CCA2 Secure Timing Attack Resistant McEliece Cryptoprocessor

Santosh Ghosh and Ingrid Verbauwhede, *Senior Member, IEEE*
{firstname.lastname}@esat.kuleuven.be

***Abstract*— This paper presents a 128-bit CCA2-secure McEliece cryptoprocessor. The existing side-channel vulnerabilities in this regard are also taken care during the implementation of such a post-quantum immune code-based cryptosystem. In order to achieve CCA2 security on original McEliece algorithm, we incorporate a SHA-3 finalist, BLAKE-512 module into the architecture. A complete binary-XGCD algorithm for Goppa field is introduced. The final design on a Virtex-6 FPGA performs an encryption in $4.74\ \mu s$ and a decryption in $0.92\ ms$. To the best of our knowledge, this is the first hardware design of McEliece with the above mentioned advanced security features which is also resistant against existing timing attacks.**

***Index Terms*— McEliece, Post-quantum Cryptography, Programmable architecture, FPGA platform, Side-channel attack.**

## 1. Introduction

MCELIECE [1] is the oldest post-quantum public key encryption scheme (PKS) based on the hardness of decoding a *general linear code* which is known to be NP-complete [3]. The original algorithm uses binary Goppa codes [4] defined on a Genus-0 curve over finite fields of characteristic 2. These codes are easy to decode with the help of a private key − thanks to an efficient algorithm due to Patterson [5]. A recent analysis on original McEliece algorithm [6] suggests to increase its key sizes by a factor of four for quantum computing. Variants of this cryptosystem exist using different types of codes. Most of them are proven less secure and are broken by structural decoding. Another fact is that the most popular asymmetric key algorithms like RSA [8], ECC [9], and Pairing [10] are easily broken once fully functional quantum computer appears [11]. Due to this fact, McEliece achieves a great importance in recent years to use in practice [12], [14], [15]. The existing results provide at most 103-bit security. Issues of physical security of McEliece cryptosystem against side-channel attacks are described in [16], [17]. However, none of the existing implementation conceals the side-channels of McEliece in practice. In this regard, the current paper contributes the followings:

- It proposes a hardware architecture for McEliece cryptosystem that provides 128-bit security.
- It integrates BLAKE-512 hash algorithm with original McEliece scheme for providing CCA2 security based on Kobara-Imai scheme [31].
- Suitable countermeasures are incorporated in the current design in order to overcome existing side-channel leakages of McEliece cryptosystem.
- To design an efficient and timing-attack resistant architecture, this paper introduces a binary-XGCD algorithm for Goppa field based on binary-GCD and its variant for computing error locator polynomials in Reed-Solomon decoding [24].
- The proposed architecture on a Virtex-6 FPGA performs encryption and decryption in $4.74\ \mu s$ and $0.92\ ms$, respectively.

Authors are with the COSIC-SCD/ESAT, KU Leuven & iMinds, Belgium. Santosh Ghosh is a Postdoctoral Fellow at KU Leuven funded by the IAP Programme P6/26 BCRYPT of Belgian Science Policy (Belspo).

In [28], it is suggested that Niederreiter cryptosystem [2] can achieve much higher performance compared to McEliece scheme. At the same time it should be noted that a huge scrambling matrix M (e.g., 163 KBits for 128-bit security [28]) must be an essential part of the private key in any case which can be omitted when using McEliece encryption. Another advantage of the McEliece scheme is that secret permutation matrix can also be omitted by deriving the control matrix $\mathbf{H}$ with permuted support. Thus, the private key size ($K_{sec}$ = $(P, M, g(z), \mathcal{L})$) of Niederreiter scheme is a bottleneck in practice, and McEliece is one step ahead in that respect.

The paper is organized with following sections. Section 2 gives an idea on McEliece algorithm. Implementation details for BLAKE-512 hash algorithm is provided in Section 3. Section 4 describes the proposed CCA2 secure McEliece encryptor followed by respective decryptor in Section 5. In Section 6, we provide the design details of underlying finite field primitives on top of which we describe design details of Goppa field primitives in Section 7. Section 8 shows the security of the current design against existing timing attacks on McEliece algorithm. Section 9 shows experimental results. Finally, the paper is concluded in Section 10.

## 2. Backgrounds

In this section we give a brief overview of McEliece PKS, classical Goppa codes, and Patterson decoding technique. We assume that the reader is familiar with the basics of error correcting codes.

### 2.1. The McEliece PKS

The McEliece PKS is named after its inventor R.J. McEliece [1]. It is a public key encryption scheme based on general coding theory. Specifically, the McEliece PKS uses Goppa codes. The strongest known attack against this scheme is based on solving the NP-hard decoding problem,

and no quantum algorithm has been developed which increases the efficiency of this attack [6]. Without presenting the mathematical foundations behind the scheme we introduce the key generation, encryption, and decryption procedures of McEliece. For these considerations, the reader is referred to [30]. It was shown that the original McEliece PKS is vulnerable against chosen-ciphertext attacks [30]. However, this problem is solved by incorporating a CCA2-conversion of the scheme [31]. The respective CCA2 variant encryption and decryption algorithms are described in § 4 and § 5.

**Bit Security and Parameter Selection.** Table I provides the respective parameters used in this paper which are suggested by Barbier and Barreto in [7] for 128-bit security with unambiguous decoding (UD).

TABLE I
PARAMETERS FOR 128-BIT SECURE MCELIECE WITH UD.

| Name | Meaning | Size (bit) |
|---|---|---|
| $m$ | Size of Galois field | 12 |
| $t$ | Number of correctable errors | 66 |
| $n$ | Code length | 3307 |
| $k$ | Code rank, $k = n - mt$ | 2515 |
| $l$ | Hash digest | 512 |
| $\mathbf{m}$ | Plaintext | 512 |
| $z$ | Ciphertext | $(n + 2k) = 4331$ |
| $R^T$ | Public key | $k \times (n - k) = 2515 \times 792$ |
| $g(x)$ | Goppa polynomial | $67 \times 12$ |
| $\mathbf{P}$ | Permutation matrix | $3307 \times 3307$ |
| $\mathbf{H}$ | Control matrix | $(66 \times 12) \times 3307$ |

**Key Generation.** The private key consists of two parts. The first part is a Goppa polynomial $g(x)$ of degree $t$ over $\mathbb{F}_{2^m}$. The second part is a randomly created $n \times n$ permutation matrix $\mathbf{P}$. The public key is generated from the private key as follows. First, compute $\mathbf{H}$ as the parity check matrix corresponding to $g(x)$. Then take $\mathbb{G}^{pub} = [\mathbb{I}_k || R]$ as the generator in systematic form corresponding to the parity check matrix $\mathbf{HP}^T$. Please note that choosing the generator in systematic form would be a security problem if the McEliece PKS was used without a CCA2-conversion. We refer the reader to [14], [15] for further details.

**Encryption.** Assume Alice wants to encrypt a message $\mathbf{m} \in \mathbb{F}_2^k$. Firstly, she has to create a random binary vector $e$ of length $n$ and Hamming weight $wt(e) = t$. Then she computes the ciphertext $z = \mathbf{m}\mathbb{G}^{pub} \oplus e$.

**Decryption.** In order to decrypt the ciphertext, Bob computes $z' = z\mathbf{P}$. He then computes the syndrome $S_{z'} = z'\mathbf{H}^T$. Afterwards, he executes an error correction algorithm with its input, the syndrome, and the permuted distorted codeword $z'$. It outputs the so-called error locator polynomial defined as:

$$\sigma(x) \quad = \quad \prod_{j \in T_{e'}} (x - \alpha_j) \in \mathbb{F}_{2^m}[x],$$

where $T_{e'} = \{i | e'_i = 1\}$ and $e'$ is the error vector of the permuted distorted code word $z$. Once the $\sigma(x)$ is known, the permuted error vector is computed as:

$$e' \quad = \quad (\sigma(\alpha_0), \sigma(\alpha_1), \cdots, \sigma(\alpha_{n-1})) \oplus (1, 1, \cdots, 1),$$

i.e., $e'_i = 1$ if $\sigma(\alpha_i) = 0$ and $e'_i = 0$ otherwise. The error vector is then found by undoing the permutation: $e = e'\mathbf{P}^T$, and the message is recovered as the first $k$ bits of $z \oplus e$, where the error correction is performed by the Patterson algorithm [5].

### 2.2. The Goppa Codes

Goppa codes [4] are a class of linear error correcting codes. The McEliece PKS makes use of irreducible binary Goppa codes which is briefly described here.

**Definition 1**. *Let the polynomial*

$$g(x) \quad = \quad \sum_{i=0}^{t} g_i x^i \in \mathbb{F}_{2^m}[x]$$

*be monic and irreducible over $F_{2^m}[x]$, and let $m, t$ be positive integers. Then $g(x)$ is called a Goppa polynomial for an irreducible binary Goppa code.*

*An irreducible binary Goppa code is then defined as:*

$$\begin{aligned} \mathcal{G}(\mathbb{F}_{2^m}, g(x)) \quad &= \quad \{c \in \mathbb{F}_2^n \mid S_c(x) \\ &:= \sum_{i=0}^{n-1} \frac{c_i}{x - \alpha_i} = 0 \bmod g(x)\}, \end{aligned}$$

*where $n \leq 2^m$, $S_c(x)$ is the syndrome of $c$, the $\alpha_i$, $i = 0, \cdots, n-1$ are pairwise distinct elements in $\mathbb{F}_{2^m}$ , and $c_i$ are the entries of the vector $c$.*

The code is defined in such a way that has length $n$, dimension $k = n - mt$ and can correct up to $t$ errors. The canonical check matrix $\mathbf{H}$ for $\mathcal{G}(\mathbb{F}_{2^m}, g(x))$ can be computed from the syndrome equation [15].

### 2.3. The Patterson Algorithm

An efficient algorithm for the determination of the error locator polynomial $(\sigma(x))$ is due to N. Patterson [5]. The algorithm exploits the following formula for computing $\sigma(x)$.

$$\sigma(x) \quad \leftarrow \quad a^2(x) + xb^2(x).$$

Defining $\tau(x) \leftarrow \sqrt{S_c^{-1}(x) + x} \bmod g(x)$ with $S_c(x)$ being the syndrome of the distorted code word $c$, the following equation holds:

$$b(x)\tau(x) \quad \equiv \quad a(x) \bmod g(x).$$

This equation can be solved by applying the Euclidean algorithm with a breaking condition concerning the degree of the remainder, assuming that no more than $t$ errors occurred [30]. Specifically, the remainder in the last step is taken as $a(x)$ and the breaking condition is $deg(a(x)) \leq \lfloor \frac{t}{2} \rfloor$. It can be shown that $deg(b(x)) \leq \lfloor \frac{t-1}{2} \rfloor$. Thus, once $a(x)$ and $b(x)$ are determined, the error locator polynomial $\sigma(x)$ is known.

## 3. IMPLEMENTATION OF BLAKE-512

Very recently, NIST announced the result of SHA-3 competition and among the five finalists, Keccak [18] has won the game. However, as per SHA-3 candidate conference [19] BLAKE [20] also was one of the strong SHA-3 finalists. No weakness of this algorithm is found till date. Additionally, the

results presented in [19] show that the mode of operation of BLAKE is provably secure and is the most flexible design with respect to hardware and software implementation [21]. It provides the best performance (213 $Kbps$ at 100 $KHz$ operating frequency) in lightweight applications ($11.3K$ Gate Equivalent (GE)) compared to other SHA-3 finalists [22]. Motivating by this fact, we explore BLAKE-512 as a secure hash algorithm as well as a pseudo random number generator (PRNG) for providing CCA2 security of original McEliece algorithm. This section presents the architecture of our BLAKE implementation.

### *3.1. Hardware Design of BLAKE-512*

The McEliece encryption and decryption algorithm have no scope to perform several independent hash operations in parallel. Thus, the chances of taking advantages of a pipelined architecture is quite low and we design a non-pipelined architecture. The BLAKE algorithm combines three previously studied components: the iteration mode HAIFA, the internal structure of the hash function LAKE, and a modified version of Bernstein's stream cipher ChaCha as compression function. It is based on three major blocks: initialization, round function, and finalization. The initialization process generates the initial state represented by sixteen 64-bit words.

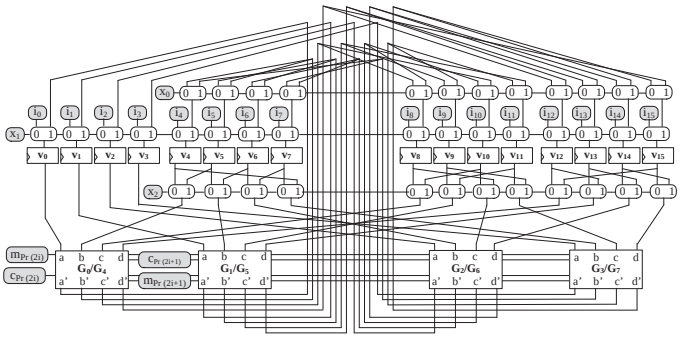


Fig. 1. Architecture of BLAKE round function.

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}$$

Four words are picked up at a time and put into the compression function $G_i$. Fig. 1 shows the overall architecture for BLAKE-512 round function which consists of 1024-bit ($16 \times 64$-bit) internal states (block). The architecture represents the right hand matrix of above initialization process by the series of $i$ vectors in row-major order. Each round of BLAKE consists of eight such $G_i$, each of which performs same operation on different inputs. Formation of $G_i$, $0 \leq i \leq 7$ is as follows:

$$G_0(v_0, v_4, v_8, v_{12});\ G_1(v_1, v_5, v_9, v_{13});\ G_2(v_2, v_6, v_{10}, v_{14});$$
$$G_3(v_3, v_7, v_{11}, v_{15});\ G_4(v_0, v_5, v_{10}, v_{15});\ G_5(v_1, v_6, v_{11}, v_{12});$$
$$G_6(v_2, v_7, v_8, v_{13});\ G_7(v_3, v_4, v_9, v_{14}).$$

At round $r$, a $G_i(a, b, c, d)$ is computed by Algorithm 1. In the algorithm, superscript indices are used to identify the inter-dependency among variables within a $G_i$ function.

**Algorithm 1** : Round function ($G$) of BLAKE-512.

**Input:** $a^{(j)}, b^{(j)}, c^{(j)}, d^{(j)}$, $m_{p_r(2i)}, m_{p_r(2i+1)}$, $c_{p_r(2i)}$, $c_{p_r(2i+1)}$.

**Output:** $a^{(j+1)}, b^{(j+1)}, c^{(j+1)}, d^{(j+1)}$.

1: $a^{(t)} \leftarrow a^{(j)} + b^{(j)} + (m_{p_r(2i)} \oplus c_{p_r(2i+1)})$;
2: $d^{(t)} \leftarrow (d^{(j)} \oplus a^{(t)}) << 32$;
3: $c^{(t)} \leftarrow c^{(j)} + d^{(t)}$;
4: $b^{(t)} \leftarrow (b^{(j)} \oplus c^{(t)}) << 25$;
5: $a^{(j+1)} \leftarrow a^{(t)} + b^{(t)} + (m_{p_r(2i+1)} \oplus c_{p_r(2i)})$;
6: $d^{(j+1)} \leftarrow (d^{(t)} \oplus a^{(j+1)}) << 16$;
7: $c^{(j+1)} \leftarrow c^{(t)} + d^{(j+1)}$;
8: $b^{(j+1)} \leftarrow (b^{(t)} \oplus c^{(j+1)}) << 11$;

Each $G_i$ function processes two message words ($m_{p_r(2i)}$, $m_{p_r(2i+1)}$) and two constant words ($c_{p_r(2i)}$, $c_{p_r(2i+1)}$), where $r$ represents the least significant decimal digit of round number (i.e., *round number* mod 10) and $p$ is a $10 \times 16$ fixed permutation matrix defined by the developers of BLAKE [20]. The functions $G_i, 0 \leq i \leq 7$ are divided in two independent sets, one consisting of $G_i, 0 \leq i \leq 3$ and other consisting of $G_j, 4 \leq j \leq 7$. The current architecture executes four $G$ functions of the first set in parallel followed by the second set. Four $G_i/G_j$-blocks shown in the architecture are responsible to execute those operations in parallel. The multiplexer-select and register-enable signals are generated by a small controller logic which is designed by means of a counter. One set of $G$ functions are executed in one clock cycle and the results are directly restored into the respective $v$ registers. Thus, one round function is performed in two clock cycles, and complete execution of 16 rounds takes 32 clock cycles.

After the rounds sequence, the new chain value $h'_0, \ldots, h'_7$ is extracted from the state $v_0, \ldots, v_{15}$ with input of the initial (previous) chain value (output of finalization) $h_0, \ldots, h_7$ and the salt $s_0, \ldots, s_3$:

$$\begin{aligned} h'_0 &\leftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8 & h'_4 &\leftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12} \\ h'_1 &\leftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9 & h'_5 &\leftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\ h'_2 &\leftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10} & h'_6 &\leftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14} \\ h'_3 &\leftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11} & h'_7 &\leftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15}, \end{aligned}$$

whereas, the first chain value is initialized by the same initial vectors of SHA-512 [20]. The finalization step consists of simple linear operations which is executed by XOR logic in a single clock cycle. In total, the execution of one BLAKE-512 takes 34 clock cycles in our design. The final chain value of a whole message is identified as the digest which is 512-bit long in case of BLAKE-512. The whole BLAKE-512 design demands 2,153 six input slices. Due to multiple 64-bit carry propagation additions, the critical delay of the G-module is 7.5 $ns$ in a Virtex-6 FPGA. That means the design can run at a maximum of 133 $MHz$ clock frequency.

## 4. Encryption Module

Encryption of a message $\mathbf{m}$ by original McEliece algorithm is realized by means of a vector matrix multiplication ($z' =$

$\mathbf{m}\mathbb{G}$) followed by an XOR ($z = z' \oplus e$) for introducing fault $e$ consisting exactly $t$ bits of one. However, in order to provide CCA2 security, McEliece encryption algorithm is extended with additional steps for introducing more indeterminacy and more randomization [14]. Algorithm 2 describes CCA2 secure McEliece encryption for which our proposed architecture is depicted in Fig. 2. The parameters selected in the algorithm are specifically for achieving 128-bit security level [7]. We use BLAKE-512 module having 512-bit digest as a secure hash unit as well as a PRNG.

**Algorithm 2** : McEliece - CCA2 secure encryption†.

**Input:** message $\mathbf{m} \in \mathbb{F}_2^l$ and public key $\mathbb{G}^{pub}$.
**Output:** ciphertext $z \in \mathbb{F}_2^{n+2l}$.

**1:** $u_1 \leftarrow$ random $(k-l)$-bit string;
**2:** $u_2 \leftarrow$ random $l$-bit string;
**3:** $\widetilde{m} \leftarrow u_1 \mid\mid H(\mathbf{m} \mid\mid u_2)$;
**4:** $c \leftarrow \widetilde{m}\mathbb{G}^{pub}$;
**5:** $e \leftarrow$ random $n$-bit error vector s. t. $wt(e) = t$;
**6:** $z_1 \leftarrow c \oplus e,\ \ z_2 \leftarrow H(u_1) \oplus \mathbf{m},\ \ z_3 \leftarrow u_2 \oplus H(e)$;
**7:** $z \leftarrow z_1 \mid\mid z_2 \mid\mid z_3$;
**8:** **return** $z$;

† parameter values: $n = 3307,\ k = 2515,\ l = 512$, and $t = 66$.

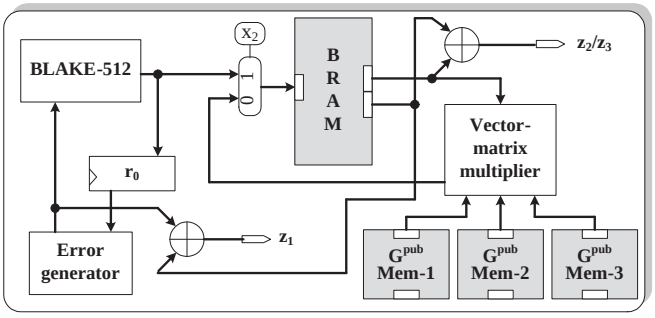


Fig. 2. Encryption unit.

The *error generator* module produces an $n$-bit random error vector $e$ having Hamming weight $wt(e) = t$, where $t = 66$ in this work. The module is realized by means of an n-bit two-way (left and right) circular shift register which is initialized by a seed $s$ having $wt(s) = t$ during system restart. There is another 256-bit register ($r_0$ shown in Fig. 2) which holds a fresh hash digest generated on-the-fly at every McEliece encryption by the BLAKE-512 core. We perform one-bit left or one-bit right shift of the first register based on each bit value of $r_0$ register. Starting from the most significant bit, if a bit value of $r_0$ is 1, then we perform a one-bit left shift, otherwise right shift. We repeat this procedure for all 256 bits of $r_0$. After 256 clock cycles (at the end of 256 random left right shifts) we lock the output as a current error vector. Therefore, based on the properties of the hash algorithm, the error vector will be random. In this regard, if we believe that BLAKE-512 is a secure hash algorithm and it generates a random pattern then from a fixed seed we get one of the $2^{256}$ error vectors randomly for every McEliece encryption. This operation is executed in parallel with other steps of the encryption. Thus, it does not add any time overhead to the McEliece encryption on the proposed cryptoprocessor.

For faster execution of vector-matrix multiplication $\widetilde{m}\mathbb{G}^{pub}$, the processing of one column is done in only one clock cycle. There are three memory modules each of which consists of 839-bit data-width that holds transposed public key ($792 \times 2515$). One column of the public key matrix is realized by the contents of three memory modules at the same address except last two bits of the third memory block, i.e., 2515-bit is realized as (839-bit $\mid\mid$ 839-bit $\mid\mid$ 837-bit, where $\mid\mid$ represents concatenation.

The CCA2 secure McEliece encryption function does invoke neither Goppa field nor $\mathbb{F}_{2^{12}}$ primitives. The main part of the encryption is Step 4 of Algorithm 2, which is realized by a binary vector-matrix multiplication. The architecture shown in Fig. 2 runs on two separate clock signals. One is primary clock (clk-1), which is the actual input clock, and other one (clk-2) is derived with twice clock period. Due to longer critical path, the BLAKE-512 module runs on clk-2, whereas all other circuit elements are operated by clk-1. Efficiency gained by two clocks are exploited for faster execution of encryption function, which is scheduled as follows:

1) BLAKE-512 is invoked four times for generating $u_1$
2) BLAKE-512 is invoked once to generate random $u_2$.
3) Execution of hash at Step 3 is scheduled immediately after concatenating message $\mathbf{m}$ with $u_2$ which too invokes BLAKE-512 for one time.
4) The hash module is invoked once more to generate a random 512-bit $r_0$ used in error generator module. At the same time the execution of Step 4 is also started.
5) When $r_0$ is available, the computation of error vector is stared. The execution of Step 4 and $H(u_1)$ are also performed in parallel.
6) The error vector is ready by the time when the execution of $H(u_1)$ completes. Thus immediately after $H(u_1)$, we schedule $H(e)$ which invokes BLAKE-512 for four times.

The respective timing graph is shown in Fig. 3. The execution of whole encryption takes 204 cycles of clk-2 followed by 796 cycles of clk-1. Except first three steps of Algorithm 2, the operations performed by the BLAKE-512 and by the error generator are hidden to the execution of $\widetilde{m}\mathbb{G}^{pub}$.

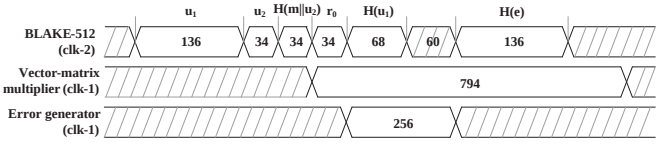


Fig. 3. Timing graph of encryption unit.

## 5. Decryption Module

McEliece decryption is a much complex and time consuming procedure than encryption. It mainly relies on decoding an erroneous Goppa code. With the help of secret permutation matrix $\mathbf{P}$ and the secret Goppa polynomial $g(x)$, the code word

is decoded and the original message is computed. Decoding is performed by Patterson algorithm [5]. In CCA2 secure version, along with original decoding technique, several hash operations are performed to ensure the integrity of message. Algorithm 3 provides step by step operations of decryption procedure adopted in the current design.

**Algorithm 3** : McEliece - CCA2 secure decryption.

**Input:** $z = (z_1, z_2, z_3) \in \mathbb{F}_2^{n+2l}$, private key $(\mathbf{P}, g(x))$, and control matrix $\mathbf{H}$.

**Output:** decrypted message $\mathbf{m} \in \mathbb{F}_2^l$ or *error*.

**1:** $c \leftarrow z_1\mathbf{P}$
**2:** $S_c(x) \leftarrow c\mathbf{H}^T(x^{t-1}, \cdots, x, 1)^T$;
**3:** $I_c(x) \leftarrow S_c^{-1}(x)$;
**4:** $\tau(x) \leftarrow \sqrt{I_c(x) + x}$;
**5:** Find $a(x)$ and $b(x)$, s.t., $b(x)\tau(x) \equiv a(x) \bmod g(x)$ and $deg(a) \leq \frac{t}{2}$;
**6:** $\sigma(x) \leftarrow a^2(x) + xb^2(x)$;
**7:** $e' \leftarrow (\sigma(\alpha_0), \sigma(\alpha_1), \cdots, \sigma(\alpha_{n-1})) \oplus (1, 1, \cdots, 1)$;
**8:** $e \leftarrow e'\mathbf{P}^T$;
**9:** $z' \leftarrow z_1 \oplus e$;
**10:** Split $z'$ to ($r$, $h$, $\widehat{m}$), having $k - l$, $l$, $n - k$ bits;
**11:** $\mathbf{m} \leftarrow z_2 \oplus H(r)$;
**12:** **if** $h = H(\mathbf{m} \mid\mid H(e) \oplus z_3)$ **then return** $\mathbf{m}$;
**13:** **else return** *"error"*;

It is considered that the private key $(\mathbf{P},\ g(x))$ and the control matrix $(\mathbf{H})$ are stored in the system memory. For practical application, a trusted platform module can be incorporated for this purpose within the proposed McEliece cryptoprocessor. The proposed decryption module is shown in Fig. 4 which executes Algorithm 3 by following way.

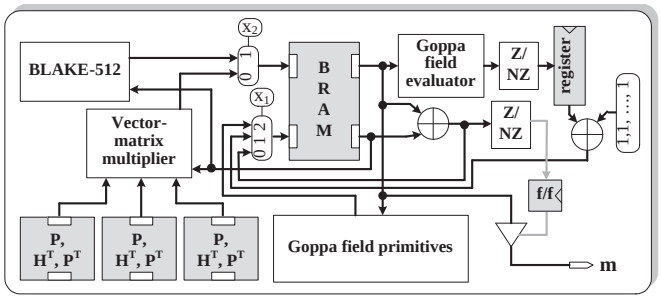


Fig. 4. Decryption unit.

- **Vector-matrix multiplication.** In the current design, we accommodate $\mathbf{P}$ and $\mathbf{P}^T$ in three memory modules having 1103-bit data-width. Each of the memory modules consists of $2^{13}$ locations. Among them, locations from 0 to 3307 are used for $\mathbf{P}$ and 4096 to 7403 are used for $\mathbf{P}^T$. In order to efficient access and compute Step 1 and Step 8 of the decryption algorithm, contents are stored with column major order in the memory modules. Thus, a full $n$-bit multiplication (between ($z_1$ or $e'$) and a column vector) and respective accumulation can be performed in one clock cycle. With this technique, the computation of $c \leftarrow z_1\mathbf{P}$ as well as $e \leftarrow e'\mathbf{P}^T$ takes $n$ clock cycles each. Similarly, in Step 2, the syndrome polynomial $S_c(x)$ is computed by $n - k$ number of clock cycles.
- **Inversion of syndrome polynomial.** Syndrome polynomial is a member of Goppa field. Inversion of this polynomial is performed on the *Goppa field primitives* module in the decryption unit shown in Fig. 4. It is performed by executing binary-XGCD (BXGCD) algorithm on Goppa field, which is described in § 7.
- **Square-root in Goppa field.** The operation $\sqrt{I_c(x) + x}$ also invokes Goppa field primitives and takes $t+1$ number of clock cycles.
- **Finding error locator polynomial $\sigma(x)$.** The polynomial $\sigma(x)$ is computed in two steps. First we invoke the Goppa field primitives for executing BXGCD with Patterson mode, for which it returns two polynomials $a(x)$, $b(x)$ such that $b(x)\tau(x) \equiv a(x) \bmod g(x)$ and $deg(a) \leq \lfloor \frac{t}{2} \rfloor$ holds. Then it executes $\sigma(x) \leftarrow a^2(x) + xb^2(x)$.
- **Finding error vector $e$ and retrieve $\mathbf{m}$.** This step is the most time consuming operation of McEliece decryption procedure. It consists of $n$ polynomial evaluations in Goppa field, each of which are performed in $t + 1$ clock cycles by the current design. Then, the actual error vector is computed through a permutation based on the private key. This procedure is realized by a vector matrix multiplication described above, which takes $n$ clock cycles.
- **Computation of CCA2 secure message.** In Step 11 the decrypted message is generated by means of four BLAKE-512 followed by an XOR. The CCA2 security check is basically realized through integrity check of the decrypted message. This is performed by first computing a hash digest for error vector (needs to invoke BLAKE-512 module for seven times) followed by computing final hash digest by invoking BLAKE-512 for a single time.

## 6. $\mathbb{F}_{2^{12}}$ Primitives

We choose $p(x) = x^{12} + x^3 + 1$ as a irreducible polynomial in $\mathbb{F}_{2^{12}}$. The addition and subtraction in this field are realized by means of simple XOR operation.

### 6.1. Multiplication in $\mathbb{F}_{2^{12}}$

The multiplication in $\mathbb{F}_{2^{12}}$ is performed by following Karatsuba technique in tower field $\mathbb{F}_{(2^4)^3}$. Algorithm 4 describes the step by step procedure which is implemented in our design. The multiplications in underlying $\mathbb{F}_{2^4}$ are performed in tower field $\mathbb{F}_{(2^2)^2}$ by Karatsuba technique. Finally, Step 4 formulates the reduction by $p(x)$. In the Algorithm 4 and rest of the paper, we use superscript $[i]$ to represent the $i$-th bit and $[i : j]$ to represent bit range from $j$ to $i$.

### 6.2. Squaring and Square-root in $\mathbb{F}_{2^{12}}$

The squaring an element $A \in \mathbb{F}_{2^{12}}$ is performed by simple bit shifting operation. Let $A = \sum a_i x^i$ then $A^2 = \sum a_i x^{2i}$, which is performed without any logic cells by simple rewiring. However, the result is reduced by executing Step 4 of Algorithm 4. The computation of $C = A^2 \in \mathbb{F}_{2^{12}}$ with irreducible

**Algorithm 4** : Multiplication and reduction in $\mathbb{F}_{(2^4)^3}$.

**Input:** $A = A_2x^8 + A_1x^4 + A_0$,
$B = B_2x^8 + B_1x^4 + B_0$, $A_i, B_i \in \mathbb{F}_{2^4}$.

**Output:** $A \cdot B$.

**1:** $M_0 \leftarrow A_0 \cdot B_0$, $T_0 \leftarrow (A_0 \oplus A_1)(B_0 \oplus B_1)$,
$T_1 \leftarrow A_1 \cdot B_1$, $T_2 \leftarrow (A_0 \oplus A_2)(B_0 \oplus B_2)$,
$T_3 \leftarrow (A_1 \oplus A_2)(B_1 \oplus B_2)$, $M_4 \leftarrow A_2 \cdot B_2$;

**2:** $M_1 \leftarrow M_0 \oplus T_1 \oplus T_0$, $M_2 \leftarrow M_0 \oplus T_1 \oplus T_2 \oplus M_4$,
$M_3 \leftarrow T_1 \oplus T_3 \oplus M_4$;

**3:** $T_4^{[22:0]} \leftarrow M_4^{[6:3]} \parallel \left(M_3^{[6:4]} \oplus M_4^{[2:0]}\right) \parallel M_3^{[3]}$
$\parallel \left(M_2^{[6:4]} \oplus M_3^{[2:0]}\right) \parallel M_2^{[3]} \parallel \left(M_1^{[6:4]} \oplus M_2^{[2:0]}\right)$
$\parallel M_1^{[3]} \parallel \left(M_0^{[6:4]} \oplus M_1^{[2:0]}\right) \parallel M_0^{[3:0]}$;

**4:** $C^{[11:0]} \leftarrow \left(T_4^{[11]} \oplus T_4^{[20]}\right)$
$\parallel \left(T_4^{[10:5]} \oplus T_4^{[22:17]} \oplus T_4^{[19:14]}\right)$
$\parallel \left(T_4^{[4:3]} \oplus T_4^{[16:15]} \oplus T_4^{[13:12]} \oplus T_4^{[22:21]}\right)$
$\parallel \left(T_4^{[2]} \oplus T_4^{[14]}\right) \parallel \left(T_4^{[1:0]} \oplus T_4^{[13:12]} \oplus T_4^{[22:21]}\right)$;

**5:** **return** $C$;

polynomial $p(x) = x^{12} + x^3 + 1$ is shown in left column of Table II. Similarly, square root of such a field element ($D = \sqrt{A} \in \mathbb{F}_{2^{12}}$) is computed by the formula shown in right column of Table II.

TABLE II
SQUARING AND SQUARE-ROOT WITH $p(x) = x^{12} + x^3 + 1$.

| **F12-SQR: $C = A^2 \in \mathbb{F}_{2^{12}}$** | **F12-SQRR: $D = \sqrt{A} \in \mathbb{F}_{2^{12}}$** |
|---|---|
| $C^{[0]} \leftarrow A^{[0]} \oplus A^{[6]}$ | $D^{[0]} \leftarrow A^{[0]} \oplus A^{[3]}$ |
| $C^{[1]} \leftarrow A^{[11]}$ | $D^{[1]} \leftarrow A^{[2]} \oplus A^{[5]}$ |
| $C^{[2]} \leftarrow A^{[1]} \oplus A^{[7]}$ | $D^{[2]} \leftarrow A^{[1]} \oplus A^{[4]} \oplus A^{[7]}$ |
| $C^{[3]} \leftarrow A^{[6]}$ | $D^{[3]} \leftarrow A^{[6]} \oplus A^{[9]}$ |
| $C^{[4]} \leftarrow A^{[2]} \oplus A^{[8]} \oplus A^{[11]}$ | $D^{[4]} \leftarrow A^{[8]} \oplus A^{[11]}$ |
| $C^{[5]} \leftarrow A^{[7]}$ | $D^{[5]} \leftarrow A^{[1]} \oplus A^{[10]}$ |
| $C^{[6]} \leftarrow A^{[3]} \oplus A^{[9]}$ | $D^{[6]} \leftarrow A^{[3]}$ |
| $C^{[7]} \leftarrow A^{[8]}$ | $D^{[7]} \leftarrow A^{[5]}$ |
| $C^{[8]} \leftarrow A^{[4]} \oplus A^{[10]}$ | $D^{[8]} \leftarrow A^{[7]}$ |
| $C^{[9]} \leftarrow A^{[9]}$ | $D^{[9]} \leftarrow A^{[9]}$ |
| $C^{[10]} \leftarrow A^{[5]} \oplus A^{[11]}$ | $D^{[10]} \leftarrow A^{[11]}$ |
| $C^{[11]} \leftarrow A^{[10]}$ | $D^{[11]} \leftarrow A^{[1]}$ |

### *6.3. Inversion in $\mathbb{F}_{2^{12}}$*

Inversion is performed by following Itoh-Tsujii technique [23]. It is based on the Fermat's little theorem and compute $a^{-1} = a^{2^{12}-2}$ with efficient powering technique. It represents $a^{-1} = a^{2^{12}-2} = [a^{2^{m-1}-1}]^2 = \prod_{j=1}^{m-1} a^{2^j}$. The computation proceeds with iterative computation of $\beta$-function. For which, let us consider $(\beta_k(a) = a^{2^k-1})_{k\in\mathbb{N}}$. That is, $\beta_0(a) = 1$, $\beta_1(a) = a$, $\cdots$, $\beta_{i+j} = \beta_i(a)^{2^j} \cdot \beta_j(a)$. Proof of this inequality is trivial. Based on above technique the chain of $\beta$-functions for $A^{-1} \in \mathbb{F}_{2^{12}}$ is shown in Table III. There is a final squaring to achieve $a^{2^{12}-2} = \beta_{11}(a)^2$. Therefore, the whole computation requires 11 squaring and 5 multiplications in $\mathbb{F}_{2^{12}}$. In the proposed design, we implement dedicated combinatorial circuits for computing $(x^2)^2$ (used in $\beta_3(a)$) and $(x^2)^5$ (used in $\beta_{11}(a)$), which is derived from squaring formula shown in Table II. The resulting architecture of our inversion unit is shown in Fig. 5. Each $\beta_i(a)$, $i = \{2, 3, 5, 6, 11\}$ computation is performed in two clock cycles, and thus, an inversion takes 10 clock cycles in the current design.

TABLE III
THE $\beta$-CHAIN FOR COMPUTING $a^{2^{11}-1}$.

| $\beta$ − **Function** | $\beta_{\mathbf{i}} \cdot \beta_{\mathbf{j}}$ | **Computation** | **Value** |
|---|---|---|---|
| $\beta_1(a)$ | − | − | $a$ |
| $\beta_2(a)$ | $\beta_{1+1}(a)$ | $\beta_1(a)^2 \cdot \beta_1(a)$ | $a^{2^2-1}$ |
| $\beta_3(a)$ | $\beta_{2+1}(a)$ | $\beta_2(a)^2 \cdot \beta_1(a)$ | $a^{2^3-1}$ |
| $\beta_5(a)$ | $\beta_{3+2}(a)$ | $\beta_3(a)^{2^2} \cdot \beta_2(a)$ | $a^{2^5-1}$ |
| $\beta_6(a)$ | $\beta_{5+1}(a)$ | $\beta_5(a)^2 \cdot \beta_1(a)$ | $a^{2^6-1}$ |
| $\beta_{11}(a)$ | $\beta_{6+5}(a)$ | $\beta_6(a)^{2^5} \cdot \beta_5(a)$ | $a^{2^{11}-1}$ |

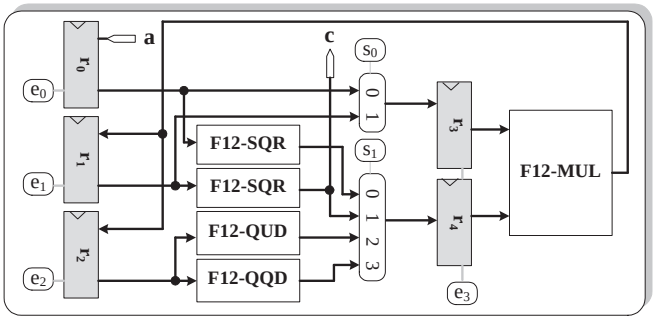


Fig. 5. The Itoh-Tsujii inversion unit.

## 7. GOPPA FIELD PRIMITIVES

The Goppa field for the current design is represented as: $g(x) = g_{66}x^{66} + g_{65}x^{65} + \cdots + g_1x + g_0$, where $g_{66} = 1$, and all other $g_i \in \mathbb{F}_{2^{12}}$, $65 \geq i \geq 0$. The parameter $x$ is a special member of the Goppa field having $g_1 = 1$ and all other $g_i = 0$. The addition and subtraction in this field are trivial and computed by simple XOR operation. However, other Goppa field operations and their respective architectural details are provided in the following sections. Overall design architecture for Goppa field primitives is shown in Fig. 6.

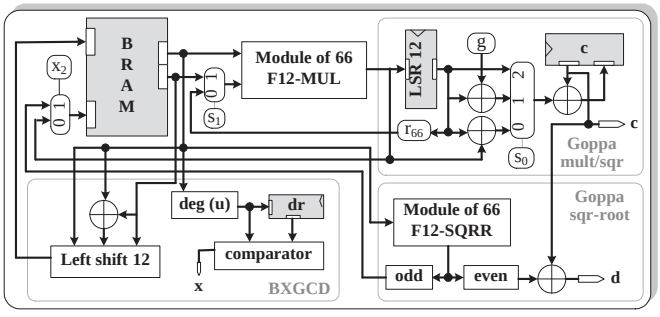


Fig. 6. The Goppa field primitive.

### *7.1. Goppa Field Polynomial Evaluation*

The proposed design evaluates a Goppa field polynomial by Horner's rule as : $b(x) = (((b_{65}x + b_{64})x + b_{63})x + \cdots + b_1)x +$

$b_0$. This evaluation procedure executes 65 multiplications and 66 additions in $\mathbb{F}_{2^{12}}$. The respective Goppa polynomial $g(x)$ is evaluated with an additional $\mathbb{F}_{2^{12}}$-addition as $g_{66} = 1$.

### 7.2. Goppa Field Multiplication and Squaring

The multiplication of two polynomials (say $a(x)$ and $b(x)$) in Goppa field is performed by following schoolbook method. In this work, the squaring is considered as multiplication with same operands. We initialize the register $c$ in the architecture shown in Fig. 6 by $b_0 a(x)$. Then for each $i$ from $i = 1$ to 65, we compute $r(x) = (b_i a(x))x \bmod g(x)$. The multiplication by $x$ is computed by means of 12-bit left shift. The reduced result of each intermediate multiplication is then accumulated (added and restore) in register $c$. The reduction (mod $g(x)$) is performed by one of the three techniques.

- If $r_{66} = 0$, then no reduction is required.
- If $r_{66} = 1$, then $r(x)$ is XORed with $g(x)$.
- If $r_{66} > 1$, then $r(x)$ is XORed with $r_{66}g(x)$, which requires another series of 66 multiplications in $\mathbb{F}_{2^{12}}$.

The proposed design consists of a module having 66 independent $\mathbb{F}_{2^{12}}$ multipliers. This module is invoked to compute operations like $t_i k(x)$, where $k(x) = \sum_{j=0}^{65} k_j x^j$ and $t_i, k_j \in \mathbb{F}_{2^{12}}$. Due to underlying parallel multipliers the whole computation of $t_i k(x)$ is completed in one clock cycle. Therefore, the computation of a Goppa field multiplication takes at most 132 clock cycles. In order to protect the design against side-channel (timing and power) attacks we enforce the design to take exactly 132 clock cycles for every Goppa field multiplications irrespective of $a(x)$ and $b(x)$. Each iteration is performed by exactly two clock cycles (one for multiplication and other for reduction). If $r_{66} \in \{0, 1\}$ then the design is enforced to compute $t_i g(x)$, with a random non-zero $t_i \in \mathbb{F}_{2^{12}}$ during the second clock.

### 7.3. Goppa Field Square-root

Goppa field square-root $\sqrt{a(x)}$ is performed by following the Huber's method [29]. It uses the precomputed constant value of $\sqrt{x}$ mod $g(x)$. It is performed as: $\sqrt{x} \equiv \sqrt{g_1(x)}(\sqrt{g_2(x)})^{-1} \bmod g(x)$, where $g(x) = g_1(x) + x g_2(x)$ with $g_1(x) = (g_0 + g_2 x^2 + g_4 x^4 + \cdots)$ and $g_2(x) = (g_1 + g_3 x^2 + g_5 x^4 + \cdots)$. The inversion $(\sqrt{g_2(x)})^{-1}$ is performed using $gcd(g(x), \sqrt{g_2(x)}) = 1$ by the BXGCD module described in the next section.

Now to compute $\sqrt{a(x)}$, let us first perform the square-root of each of the coefficients $a_i$, $0 \leq i \leq 65$ of $a(x)$, and divide in odd and even sets based on the value of $i$. The weight of each of the coefficient will now become half, i.e., $x^{\lfloor i/2 \rfloor}$. Thus if $i$ is odd, then the respective element in the odd set is multiplied with $\sqrt{x}$. Finally, we take into account the elements of both sets w.r.t their weights to form the resultant polynomial as a member of Goppa field. There are 33 elements in the odd set, which forms a polynomial $o(x)$ of degree 32. For computing the multiplication $\sqrt{x} \cdot o(x)$, the module of 66 parallel multipliers is invoked for 65 times to perform multiplication and reduction as described in the previous section. In total, the latency of a square-root operation in Goppa field is 68 clock cycles on the proposed design.

### 7.4. Goppa Field BXGCD

We develop Algorithm 5 as binary XGCD (BXGCD) for Goppa field. The algorithm is flexible to compute one of the following three operations which are identified by mode input.

- Greatest common divisor (gcd) of two polynomials.
- Polynomial inversion.
- Polynomial decomposition.

Two most difficult operations in the BXGCD algorithm are finding the degree of a polynomial ($deg(u)$) and computing the multiplication like $a_i b$, where $a_i \in \mathbb{F}_{2^{12}}$ and $b$ is a member of Goppa field. The latter one is performed on the module of 66 parallel multipliers which is shared by other Goppa field primitives. Whereas, we develop a dedicated fully combinatorial circuit for finding out the degree of a Goppa field polynomial. It finds out the first nonzero coefficient of an input polynomial ($a$) starting from the most significant coefficient ($a_{65}$).

**Algorithm 5** : Binary-XGCD for Goppa field.

**Input:** $a, g$, and mode $m = \{pat,\ gcd,\ inv\}$.
**Output:** gcd$(a, g)$, $a^{-1}$, or $\alpha$ *and* $\beta$.

**1:** $u \leftarrow a,\ \ v \leftarrow g,\ \ \rho \leftarrow 1,\ \ \varrho \leftarrow 0;$
**2:** **while** $(u \neq 0)$ and $(v \neq 0)$ **do**
**3:** **while** $x$ divides $u$ **do**
**4:** $u \leftarrow u/x;$
**5:** **if** $x$ divides $\rho$ **then** $\rho \leftarrow \rho/x;$
**6:** **else** $\rho \leftarrow (g_0\rho + \rho_0 g)/x;$
**7:** **end while**
**8:** **while** $x$ divides $v$ **do**
**9:** $v \leftarrow v/x;$
**10:** **if** $x$ divides $\varrho$ **then** $\varrho \leftarrow \varrho/x;$
**11:** **else** $\varrho \leftarrow (g_0\varrho + \varrho_0 g)/x;$
**12:** **end while**
**13:** **if** $deg(u) > deg(v)$ **then**
**14:** $u \leftarrow v_0 u + u_0 v,\ \ \rho \leftarrow \varrho_0\rho + \rho_0\varrho;$
**15:** **else**
**16:** $v \leftarrow v_0 u + u_0 v,\ \ \varrho \leftarrow \varrho_0\rho + \rho_0\varrho;$
**17:** **if** $(deg(u) < 33)$ and $(m = pat)$ **then**
**18:** **return** $\alpha \leftarrow u,\ \beta \leftarrow \rho;$
**19:** **end while**
**20:** **if** $(u = 0)$ **then**
**21:** **if** $m = gcd$ **then return** $v;$
**22:** **else if** $m = inv$ **then return** $\varrho;$
**23:** **else**
**24:** **if** $m = gcd$ **then return** $u;$
**25:** **else if** $m = inv$ **then return** $\rho;$

The main motivation to implement the BXGCD is two folds. First, it avoids the polynomial division. Second, it is easy to implement as a side-channel attack resistant hardware module. It is shown in [17], [16], [27] that the timing difference for $S_c(x)^{-1}$ in Patterson algorithm can be exploited to break McEliece cryptosystem which is overcome in our design.

## 8. Security Against Timing Based SCA

Side-channel vulnerabilities of the McEliece cryptosystem were studied in [16], [17], [25], [26], [27]. Except [25], other

works listed above aims to find weakness of the McEliece against timing variations for computing different operations in the decryption procedure. The current design for the first time takes care of these SCA-vulnerabilities. Here we give a brief discussion on the identified weaknesses applicable to hardware implementation of the McEliece, and the respective countermeasures are embedded in the current design.

### *8.1. Timing Attack on Error Locator Polynomial [16]*

This attack exploits the fact that the error locator polynomial $\sigma(x)$ essentially has degree equal to the error bits and is at most $t$. The polynomial $\sigma(x)$ is computed from ciphertext $z$ (without CCA2), which is a part of ciphertext $z_1$ in CCA2 version. The attack believes that the time taken for evaluating Step 7 of Algorithm 3 is increased with the degree of $\sigma(x)$. Remember that this is the most time consuming operation in McEliece decryption algorithm. The attack flips a bit of $z_1$ and measures the execution time. Essentially, if there is an error in the respective $i$-th bit of $z_1$ then it is now cleared with bit flip which effectively generates a $\sigma(x)$ with degree $t-1$ instead of $t$. Thus the time taken in this decryption will be less than original $z_1$.

However in the current design, the evaluation procedure is based on Horner scheme as described in § 7. It is implemented in such a way that irrespective of the $x$ and the polynomial-coefficients (whether they are zero or non-zero), it executes 65 multiplications and 66 additions in $\mathbb{F}_{2^{12}}$ for evaluating any polynomial in the respective Goppa fields. Therefore, the execution of Step 7 takes same amount of time for all ciphertexts, which ensures its security against the attack presented above.

### *8.2. Timing Attack on $\tau(x)$ to $\sigma(x)$ Computation [17]*

In this attack, the author examines the timing variations for computing $\sigma(x)$ from $\tau(x)$ in Patterson decoding. This is performed by two steps : invoking XGCD (Step 5) and performing two squarings (Step 6) in Algorithm 3. Based on the previous attack, it exploits the fact that the execution time of XGCD algorithm for Patterson varies with the degree of $a(x)$ and $b(x)$, which are related to the number of error bits of $z_1$. This attack also exploits the timing variations for executing squarings of $a(x)$ and $b(x)$ in Step 6.

The current design implements the binary-XGCD algorithm in such a way that it executes Patterson with a fixed iteration counts irrespective of $\tau(x)$. Every iterations are also executed with a fixed number of clock cycles which ensures no timing variation with processed data. The squarings in Step 6 are performed by invoking *Goppa mult/sqr* module shown in Fig. 6. It executes every Goppa field multiplications and squarings in fixed clock cycles as described in § 7.2. However, it may be argued that the degree of $a(x)$ and $b(x)$ in Patterson $\simeq \frac{t}{2}$ and the squaring time will be around half compared to a squaring of degree $t$ polynomial. This savings of cycles count ($t$ or 66 cycles) is less than $0.1\%$ of total decryption time. Thus, our design makes almost zero performance loss by executing two squarings in Step 6 as full Goppa field polynomial squarings which rather makes the design secure against above timing attack.

### *8.3. SCA on the Goppa Polynomial [27]*

This attack aims at side-channel leakages from Patterson's algorithm for finding out secret Goppa polynomial. It primarily exploits the Huber's square-root computation technique as described in § 7.3. The attacker tries to find out the fixed polynomial $\sqrt{x} \bmod g(x)$ when it is performed as: $\sqrt{x} \equiv \sqrt{g_1(x)}(\sqrt{g_2(x)})^{-1} \bmod g(x)$. After getting $\sqrt{x}$, attacker performs $y(x) = (\sqrt{x} \bmod g(x))^2 + x = z(x)g(x)$ and obtains $g(x)$ by factoring $y(x)$. This attack on Huber's method becomes more easy if the attacker can choose a syndrome such that $I_c(x) = S_c^{-1}(x) \bmod g(x)$ has all coefficients $I_{2i+1}$ set to zero. In case of general $I_c(x)$, the authors suggested a power analysis with a Hamming weight leakage model for finding out $\sqrt{x} \bmod g(x)$. Therefore, success of the attack depends on the implementation of Goppa field multiplier.

We implement Goppa field multiplication by school-book method with the help 66 parallel $\mathbb{F}_{2^{12}}$-multipliers as described in § 7.2. Goppa polynomial is stored in BRAM, from which all 66 coefficients are accessed in parallel and used them to perform intermediate reduction. Thus, the major assumption of the attack as described in § 4.0.5, [27] – "learn the position of coefficients in log/anti-log LUTs" is no longer valid in our implementation. It ensures that current implementation is secure against above vulnerability. The existing fault attack described in [27] is not considered in the current work as it is specifically for software implementations.

## 9. Implementation Results

The design has been done in Verilog (HDL) on Xilinx ISE Design Suite 12.4. Table IV shows the implementation results. The area requirement for individual operations shown in the table indicates the total resources utilized to execute respective operation. On a Virtex-6 xc6vhx255t-3ff1155 FPGA, one 128-bit CCA2 secure McEliece encryption finishes in $4.74\ \mu s$ and respective decryption in $0.92\ ms$ at 254 MHz. In total, as shown in Table V, the top level design including BLAKE-512, $\mathbb{F}_{2^{12}}$-primitives, Goppa primitives, encryption, and decryption blocks uses $5,357$ logic slices and 488 BRAMS on above mentioned Virtex-6 FPGA.

TABLE IV
Implementation results

| Operation | Area | | Clock [MHz] | Cycle counts | Time |
|---|---|---|---|---|---|
| | Slice | BRAM | | | |
| **BLAKE-512** | 2,153 | – | 133 | 34 | 0.26 $\mu s$ |
| $\mathbb{F}_{2^{12}}$**-Mult.** | 28 | – | – | – | 2.00 $ns$ |
| $\mathbb{F}_{2^{12}}$**-Sqr.** | 5 | – | – | – | 0.77 $ns$ |
| $\mathbb{F}_{2^{12}}$**-Sqrt.** | 5 | – | – | – | 0.77 $ns$ |
| $\mathbb{F}_{2^{12}}$**-Inv.** | 67 | – | 447 | 10 | 0.02 $\mu s$ |
| **Goppa-Eval.** | 60 | – | 445 | 67 | 0.15 $\mu s$ |
| **Goppa-M/S.** | 2,315 | – | 333 | 133 | 0.40 $\mu s$ |
| **Goppa-Sqrt.** | 2,123 | – | 333 | 68 | 0.20 $\mu s$ |
| **Goppa-Inv.** | 2,567 | 11 | 254 | 1,584 | 6.24 $\mu s$ |
| **McEliece-Enc.** | 2,297 | 71 | 254 | 1,203 | 4.74 $\mu s$ |
| **McEliece-Dec.** | 5,163 | 417 | 254 | 232,514 | 0.92 $ms$ |

Cycle counts shown in Table IV are calculated with respect to the primary clock used in the respective module. In the CCA2-secure McEliece encryption and decryption modules, the hash

TABLE V
COMPARISON OF THE MCELIECE IMPLEMENTATIONS

| Properties | **Our Design** | Cryptoprocessor [14] | Co-processor [13] | | MicroEliece [12] |
|---|---|---|---|---|---|
| **Bit Security** | 128-bit | 103-bit | 80-bit | | 80-bit |
| **CCA2 Security** | √ | √ | × | | × |
| **SCA Resistant** | √ | × | × | | × |
| **Platform** | Virtex-6 | Virtex-5 | Virtex-5 | Spartan-3an | Spartan-3an |
| **Slices** | 5,357 | 14,537 | 1385 | 2979 | 11,218 |
| **BRAM** | 488 | 75 | 5 | | 20 |
| **Clock Frequency** | 254 MHz | 163 MHz | 190 MHz | 92 MHz | 85 MHz |
| **Clock Cycles for Encryption** | 1,203 | 81,500 | – | | 161,480 |
| **Encryption Latency** | 4.74 $\mu$s | 0.50 ms | – | | 1.07 ms |
| **Clock Cycles for Decryption** | 232,514 | 210,270 | 94,249 | | 891,736 |
| **Decryption Latency** | 0.92 ms | 1.29 ms | 0.50 ms | 1.02 ms | 10.82 ms† |

† Best possible latency assuming that an ideal PRNG to generate $S^{-1}$ is used.

functions on BLAKE-512 module are performed by a derived clock having half frequency. However in the result table, it is counted in terms of primary clock which is twice faster than the original clock of BLAKE-512 module.

### *9.1. Comparison*

Table V evaluates the performance of our proposed McEliece cryptoprocessor with existing designs. Implementations are targeted at different platforms with unequal parameters (128, 103, 80 bit). Thus, they are not directly comparable. As mentioned in § 2, the chosen parameters $\{n = 3307, k = 2515, t = 66\}$ in our implementation is used for the first time that achieves a superior (128-bit) security compared to all related designs. Though the only existing design in [14] considers CCA2-security, but along with this, ours is the first design which also provides security against existing side-channel vulnerabilities. The high speed of the current design is due to its parallel datapath and respective memory configuration providing high bandwidth. Hardware sharing technique adopted in our architecture for computing several operations help to realize the complete datapath with relatively less slices. For example, we perform the permutation of the error vector $e$ on vector-matrix multiplier without implementing additional root searching hardware to follow Horner scheme as implemented in [13]. Though the memory requirements of the design is slightly high but it is less than the available BRAM of the chosen FPGA. Apart from the memory requirement, the slice count of the proposed McEliece cryptoprocessor is relatively low, which means it can fit into a lower footprint FPGA with online transfer of $\mathbf{P}$ and $\mathbf{H}$ matrices during decryption. Only bottleneck in this approach is that the decryption cannot be started until $\mathbf{P}$ and $\mathbf{H}$ are transmitted. This overhead time heavily increases the run-time of the proposed design. However, transmission of $\mathbf{P}$ as well as permutation $z_1\mathbf{P}$ and unpermutation $e'\mathbf{P}^T$ in the decryption (Algorithm 3) procedure could be eliminated by computing the syndrome $S_{z_1} = z_1\mathbf{H}^T$, where $\mathbf{H}$ is derived from a permuted support. This will reduce the transmission time of an $n \times n$ bit matrix $\mathbf{P}$ and the execution time of two vector-matrix multiplications each of which takes $n$-clock cycles in the current design.

## 10. CONCLUSION

This work proposed a cryptoprocessor for computing McEliece post-quantum PKS. For the first time, a timing-attack resistant architecture is designed for 128-bit CCA2 secure McEliece. The current design exploits the advantages of a new cryptographic hash function, a SHA-3 finalist BLAKE. One more important outcome of this paper is that it introduced a complete binary-XGCD algorithm for Goppa field applicable to McEliece cryptosystem. The techniques shown in this paper will help to develop similar cryptoprocessor for other code-based cryptosystems like Niederreiter PKS.

## ACKNOWLEDGMENTS

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy) and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

## REFERENCES

[1] R.J. McEliece, "A public-key cryptosystem based on algebraic coding theory," *JPL DSN Progress Report*, pp. 114–116, 1978.
[2] H. Niederreiter, "Knapsack-Type Cryptosystems and Algebraic Coding Theory," Problems of Control and Information Theory 15, pp. 159–166, 1986.
[3] E. Berlekamp, R. McEliece, and H. VanTilborg, "On the inherent intractability of certain coding problems," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386 , May 1978.
[4] V.D. Goppa. "A new class of linear correcting codes," *Problems of Information Transmission*, vol. 6, pp. 207–212, 1970.
[5] N. Patterson, "Algebraic decoding of Goppa codes," *IEEE Trans. Information Theory*, vol. 21, no. 2, pp. 203–207, Mar. 1975.
[6] D.J. Bernstein, "Grover vs. McEliece," PQCrypto '10, LNCS 6061, pp. 73–80, 2010.
[7] M. Barbier and P.S.L.M. Barreto, "Key reduction of McEliece's cryptosystem using list decoding," ISIT 2011, IEEE, pp. 2681–2685, 2011.
[8] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Comm. ACM*, vol. 21, pp. 120–126, 1978.
[9] N. Koblitz, "Elleptic curve cryptosystems," *Math. Computation*, vol. 48, pp. 203–209, 1987.
[10] D. Boneh and M.K. Franklin, "Identity-based encryption from the Weil pairing," Crypto '01, LNCS 2139, pp. 213–229, 2001.
[11] P.W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," FOCS '94, pp. 124–134, 1994.
[12] T. Eisenbarth, T. Güneysu, S. Heyse, and C. Paar, "MicroEliece: McEliece for embedded devices," CHES '09, LNCS 5747, pp. 49–64, 2009.

[13] S. Ghosh, J. Delvaux, L. Uhsadel, and I. Verbauwhede, "A speed area optimized embedded co-processor for McEliece cryptosystem," IEEE ASAP 2012, Delft, Netherlands, 2012.
[14] A. Shoufan, T. Wink, H.G. Molter, S.A. Huss, and E. Kohnert, "A novel cryptoprocessor architecture for the McEliece public-key cryptosystem," *IEEE Trans. on computers*, vol. 59, no. 11, pp. 1533–1546, Nov. 2010.
[15] F. Strenzke. "A smart card implementation of the McEliece PKC," WISTP '10, LNCS 6033, pp. 47–59, 2010.
[16] F. Strenzke, E. Tews, H.G. Molter, R. Overbeck, and A. Shoufan, "Side channels in the McEliece PKC," PQCrypto '08, LNCS 5299, pp. 216–229, 2008.
[17] A. Shoufan, F. Strenzke, H.G. Molter, and M. Stöttinger, "A timing attack against Patterson algorithm in the McEliece PKC," ICISC '09, LNCS 5984, pp. 161–175, 2009.
[18] G. Bertoni, J. Daemen, M. Peeters, and G.V. Assche1, "The Keccak sponge function family," http://keccak.noekeon.org/
[19] D. Dodson, "The third SHA-3 candidate conference. Computer security division," NIST, http://csrc.nist.gov/groups/ST/hash/sha–3/Round3/March2012/index.html
[20] J.P. Aumasson, L. Henzen, W. Meier, and R. Phan, "SHA-3 proposal BLAKE (version 1.3)," 2009, [online] available at http://www.131002.net/blake.
[21] E. Andreeva, A. Luykx, and B. Mennink, "Provable security of BLAKE with non-ideal compression function," IACR eprint archive, report 2011/620, http://eprint.iacr.org/
[22] E.B. Kavun and T. Yalcin, "On the suitability of SHA-3 finalists for lightweight applications," presented at: The Third SHA-3 Candidate Conference, 2012. [Online] csrc.nist.gov/groups/ST/hash/sha-3/Round3/March2012/documents/papers/KAVUN_paper.pdf
[23] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF($2^m$) using normal bases," *Inf. Comput.*, vol. 78, no. 3, pp. 171–177, 1988.
[24] F. Arguëllo, "Binary GCD algortihm for computing error locator polynomials in Reed-Solomon decoding," *Electronics Letters*, IEE, Vol. 41, No. 13, p. 2, June 2005.
[25] S. Heyse, A. Moradi, and C. Paar, "Practical power analysis attacks on software implementations of McEliece," PQCrypto 2010, LNCS 6061, pp. 165–181, 2010.
[26] F. Strenzke, "A timing attack against the secret permutation in the McEliece PKC," PQCrypto 2010, LNCS 6061, pp. 95–107, 2010.
[27] R. Avanzi, S. Hoerder, D. Page, and M. Tunstall, "Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems," *J. Cryptographic Engineering*, vol. 1, No. 4, pp. 271–281, 2011.
[28] S. Heyse and T. Güneysu, "Towards one cycle per bit asymmetric encryption: code-based cryptography on reconfigurable hardware," CHES 2012, LNCS 7428, pp. 340–355, 2012.
[29] K. Huber, "Note on decoding binary Goppa codes," *Electronics Letters*, vol. 32, no. 2, pp. 102–103, 1996.
[30] D. Engelbert, R. Overbeck, and A. Schmidt, "A summary of McEliece-type cryptosystems and their security," *Journal of Mathematical Cryptology*, vol. 1, no. 2, pp. 151–199, 2007.
[31] K. Kobara and H. Imai, "Semantically secure McEliece public-key cryptosystems − conversions for McEliece PKC −," PKC 2001, LNCS 1992, pp. 19–35, 2001.

**Santosh Ghosh** obtained his B.Tech degree in Computer Science and Engineering from Haldia Institute of Technology, Haldia, WB, India in 2002. He obtained his M.S. and Ph.D. in 2008 and 2011 from the Department of Computer Sc. and Engg., Indian Institute of Technology Kharagpur, India. Currently, he is a Post-doctorate Researcher at COSIC-SCD/ESAT, KU Leuven, Belgium. His research interests include cryptography and network security, VLSI design of cryptosystems, side-channel analysis, and secure design-for-test infrastructure. He has authored about 8 Journal and more than 25 Conference papers and has served as Reviewers of several International Conferences and Journals.

**Ingrid Verbauwhede** received the electrical engineering degree and PhD degree from the KU Leuven, Belgium, in 1991. From 1992 to 1994, she was a postdoctoral researcher and visiting lecturer at the University of California, Berkeley. From 1994 to 1998, she worked for TCSI and ATMEL in Berkeley, California. In 1998, she joined the faculty of University of California, Los Angeles (UCLA). She is currently a professor at the KU Leuven and an adjunct professor at UCLA. At KU Leuven, she is a co-director of the Computer Security and Industrial Cryptography (COSIC) Laboratory. Her research interests include circuits, processor architectures and design methodologies for real-time embedded systems for security, cryptography, digital signal processing, and wireless communications. This includes the influence of new technologies and new circuit solutions on the design of next-generation systems on chip. She was the program chair of CHES '07, CHES '12, ASAP '08, ISLPED '02. She was also the general chair of ISLPED '03. She was a member of the executive committee of DCA '05 and DAC '06. She is a senior member of the IEEE.