Towards a Secure Goppa Decoder in Hardware Implementation

Tania Richmond, Pierre-Louis Cayrel, Viktor Fischer, Pascal Véron

Towards a Secure Goppa Decoder in Hardware Implementation

Tania RICHMOND[1], Pierre-Louis CAYREL[1], Viktor FISCHER[1], Pascal VÉRON[2]

[1] Laboratoire Hubert CURIEN,
Rue du Prof. Benoit LAURAS, 18,
42000, Saint-Étienne, France.
{tania.richmond,pierre.louis.cayrel,
fischer}@univ-st-etienne.fr
[2] Institut de Mathématiques
B.P. 20132,
83957, La Garde, France.
veron@univ-tln.fr

Abstract

The irreducible binary Goppa codes are widely used in code-based cryptography, like in the McEliece cryptosystem. The aim of this work is to design an efficient and secure hardware implementation of a Goppa decoder. We will show how to adapt a common step of all decoding algorithms to obtain a "leakage resistant" variant.

1 Introduction

There exists several domains in cryptography. Code-based cryptography has interesting properties: Fast computations, a well theorical complexity and no quantum algorithm in polynomial time breaks the underlying NP-hard problem [4].
The first public-key cryptosystem (PKC) based on error-correcting codes was proposed by McEliece in 1978 in [11]. In his paper, he proposed to use the family of Goppa codes. Permuted Goppa codes present some advantages: they look like random codes, have at least one efficient decoding algorithm, and it is a dense family of codes. Goppa decoder can be used as a trapdoor in code-based cryptography, both for encryption scheme (like McEliece public-key encryption scheme [11]) and signature scheme (like CFS signature scheme [5]).
It is in this context that takes place our work. We are interested in the way that Goppa decoding algorithms are translated into circuits in hardware. Furthermore, we propose a partial hardware implementation of them.
The remainder of this paper is organized as follows. In Section 2, we will recall some preliminaries about the McEliece PKC and the Goppa codes. In Section 3, we will briefly present the principle of side-channel attacks. In Section 4, we will explain our hardware implementation of a partial Goppa decoder. Finally, we will conclude this paper in Section 5.

2 McEliece PKC with Goppa codes

The reader who wants more explanations about the McEliece PKC or the Goppa codes can find them in Sections 3 and 4 in [7] or directly in [11, 6, 3].

2.1 McEliece PKC

The McEliece PKC is composed of the three following algorithms.
Key generation: In inputs of this algorithm, we have two integers m and t, respectively for extension degree of Galois field and Goppa polynomial degree. The steps are:

1. Choose a t-error correcting Goppa code $\mathcal{C}$ with parameters $[n, k]$ (cf. Subsection 2.2).
2. Compute a $k \times n$ generator matrix G of $\mathcal{C}$.
3. Randomly choose a non-singular binary $k \times k$ matrix S and a $n \times n$ permutation matrix P and compute the generator matrix $\tilde{G} = S \cdot G \cdot P$ and the inverse matrices S^{-1} and P^{-1}.
4. Return the public key $pk = (\tilde{G}, m, t)$ and the secret key $sk = (\mathcal{C}, S^{-1}, P^{-1})$.

Encryption: In inputs of this algorithm, we have the public key $pk = (\tilde{G}, m, t)$ and a message M. In output, this algorithm returns the ciphertext C. The steps are:

1. Randomly choose an error-vector E of weight t.
2. Compute $C = M \cdot \tilde{G} \oplus E$.
3. Return C.

Decryption: In inputs of this algorithm, we have the secret key $sk = (\mathcal{C}, S^{-1}, P^{-1})$ and a ciphertext C. In output, this algorithm returns the plaintext M. The steps are:

1. Compute $\tilde{C} = C \cdot P^{-1}$.
2. Decode $\tilde{C}$ with a Goppa decoder to obtain $\tilde{M} = M \cdot S$.
3. Compute $M = \tilde{M} \cdot S^{-1}$.
4. Return M.

2.2 Goppa codes

Definition: In this paper, we are only interested in irreducible binary Goppa codes. A Goppa code $\mathcal{C}$ is a linear code generated by the Goppa polynomial g and the support $\mathcal{L}$. g is a monic irreducible polynomial of degree t. $\mathcal{L} = \{\alpha_1, \ldots, \alpha_n\}$ is a subset of n elements of the Galois field $\mathbb{F}_{2^m}$ with $g(\alpha_i) \neq 0$ $\forall i \in [\![1, n]\!]$. $\mathcal{C}$ is the set of words c of length n which verify the following equivalence:
$-\sum_{i=1}^{n} \frac{c_i}{g(\alpha_i)} \times \frac{g(x)-g(\alpha_i)}{x-\alpha_i} \equiv 0 \mod g(x)$.
Decoding: The aim of a Goppa decoder is to solve the key equation as follows: Find the so-called error locator polynomial σ such that $S_C(x) \times \sigma(x) = \sigma\prime(x) \mod g(x)$. All σ roots correspond to all error positions. That is why, to decode a codeword C into a message M, there are the three following steps:

1. Compute the syndrome polynomial: $S_C(x) = -\sum_{i=1}^{n} \frac{C_i}{g(\alpha_i)} \times \frac{g(x)-g(\alpha_i)}{x-\alpha_i}$.
2. Solve the key equation to obtain the error locator polynomial: $\sigma(x)$.
3. Find the roots $\beta_i \in \mathcal{L}$ of the error locator polynomial: $\sigma(x) = \prod_{i=1}^{t} (x - \beta_i)$.

The second step can be executed by extended Euclidean algorithm, Berlekamp-Massey algorithm [10] or Patterson algorithm [13].

3 Side-channel attacks

Principle. The principle of side-channel attacks appeared only in 1996 [9]. A side-channel attack is an attack which exploits the laws of the physics to obtain some information contained in channels associated to an implementation, a circuit. The purpose is to extract secrets manipulated by smart cards or cryptographic components. The side-channel does not aim at transmitting voluntarily some information. It leaks information that an observer can interpret. On the other hand, algorithmic or physic countermeasures can be proposed to stop such a leakage. Implementation of a cryptosystem is a tradeoff between security and efficiency. That is why cryptanalysis and implementation must be simultaneously considered.

Examples. The main side-channel attacks are: timing attack and power consumption attack. The first one exploits variations of processing time of a program, which depends on the size of the data; the second one the power consumption of the whole system in an immediate way, which depends on treatments made on the data.
Study of the McEliece cryptosystem resistance against this type of attacks began only five years ago [17]. There exists several side-channel attacks and associated countermeasures [2, 12, 14, 15, 16, 17], we want to provide an implementation secure faces all those attacks. In order to reach this goal, we implemented the most critical step in a Goppa decoder (the root-finding step). Our aim is to find new vulnerabilities, namely those related to information leakages via power consumption.

4 Hardware implementation of a vulnerable part of the McEliece decryption algorithm

As explained above, our final objective is to implement cryptographic protocols based on error-correcting codes on various supports and then to analyze the behavior of these implementations against side-channel attacks.
First of all, we needed to localize the part of the McEliece algorithm, which could be interesting for the attacker. We believe that one of the most important in the McEliece PKC is the decryption, in which the hardware manipulates secret key. We therefore concentrated our effort to this part of the McEliece PKC, which can be implemented in three ways: using the Patterson [13], Berlekamp-Massey [10] or extended Euclidean algorithm. We selected the so-called error locator polynomial, which is used during the Goppa code decoding, as the most security critical operation.
Our next task was to implement it in hardware, in order to be able to observe later dynamically its power consumption, which is related to confidential data being processed. Once the information leakage point will be found, our next objective will be to propose an efficient countermeasure on either algorithmic or implementation level or both.

4.1 Implementation of Galois field multiplier in hardware

The main component of all decryption algorithms is the Galois field multiplier. Therefore, we started our work by designing an efficient architecture of the multiplier block.

Algorithm of multiplication in Galois fields:
We decided to use the next Horner's scheme [8] of multiplication algorithm.

Description: Multiplication of two elements α and β of a finite field $\mathbb{F}_{2^m} = \mathbb{F}_2[X]/Q_m(X)$. The product denoted r by Galois Multiplier (GM) is in $\mathbb{F}_{2^m}$.
Parameters: m the degree of the polynomial,
$Q_m(X) = \sum_{i=0}^{m} q_i X^i$ a polynomial of degree m on $\mathbb{F}_2$ (Version 2).
Inputs: Two elements of $\mathbb{F}_{2^m}$ represented by two polynomials of degree $m-1$:
$\alpha(X) = \sum_{i=0}^{m-1} \alpha_i X^i$ and $\beta(X) = \sum_{i=0}^{m-1} \beta_i X^i$, where $\alpha_i, \beta_i \in \{0,1\}$,
$Q_m(X) = \sum_{i=0}^{m} q_i X^i$ a polynomial of degree m on $\mathbb{F}_2$ (Version 1)
Output: Product of the inputs seen as a polynomial of degree $m-1$: $r(X) = \sum_{i=0}^{m-1} r_i X^i$.

We can represent all polynomials as vectors, as follows:

$$\alpha = \sum_{i=0}^{m-1} \alpha_i X^i$$
$$\Rightarrow \alpha = (\alpha_{m-1}, \ldots, \alpha_0).$$

GM:
$r(X) \leftarrow \alpha_{m-1}\beta(X)$
For i **from** $m-1$ **downto** 1 **do**
  $r(X) \leftarrow r(X)X + \alpha_{i-1}\beta(X) + r_{m-1}Q_m(X)$
End for
Return r

Example: $m = 5$
$Q_m(X) = x^5 + x^3 + x^2 + x + 1$
$\alpha(x) = x^4 + x^2 + x$ and $\beta(x) = x^4 + x$

	x^5	x^4	x^3	x^2	x	1
Q_m	1	0	1	1	1	1
α		1	0	1	1	0
β		1	0	0	1	0
r		1	0	0	1	0
$(i = 4)$	1	0	0	1	0	0
		0	0	0	0	0
	1	0	1	1	1	1
r	0	0	1	0	1	1
$(i = 3)$				⋮		

Implementation of a parameterized Galois field multiplier:
We implemented two versions of the Galois field multiplier algorithm from the preceding section in VHDL. The size of the multiplier is very flexible, since m is given as a parameter. Version 1 of the multiplier block has the polynomial given as an input (a binary vector specifying coefficients of the polynomial) and the second version (which is less flexible, but also less expensive) has the polynomial given as a second parameter. We implemented both versions on the Evariste II FPGA [1] module featuring Altera Cyclone III device EP3C25F256-C8. Implementation results obtained using Altera Quartus II version 9.1 software are presented in Table 1 in term of logic Altera cells (LCELLs) and frequency (freq). As it can be seen (and expected), the flexible solution using polynomial as an input (Version 1) is always slower and larger. It is therefore important to evaluate if the selected decryption algorithm needs such flexibility or not.

Version 1 (with variable Q_m)			Version 2 (with fixed Q_m)		
m	LCELLs	freq (MHz)	m	LCELLs	freq (MHz)
5	35	348	5	21	453
6	54	275	6	29	405
7	68	207	7	37	455
8	90	175	8	56	355
9	117	172	9	59	414
10	140	135	10	83	321
11	178	140	11	87	411
12	209	118	12	123	276

Table 1: Implementation results for Version 1 (with variable Q_m) vs Version 2 (with fixed Q_m)

4.2 Implementation of error locator polynomial in hardware

We implemented two versions of architectures for error vector computation. The first one computes the polynomial σ from right to left [R2L] and the second one from left to right [L2R] (according to the Horner algorithm [8]). The polynomial has the following forms:

$$\sigma(X) = \sigma_t X^t + \sigma_{t-1} X^{t-1} + \ldots + \sigma_1 X + \sigma_0 \text{ [R2L]}$$
$$\sigma(X) = (((\sigma_t X + \sigma_{t-1})X + \ldots)X + \sigma_1)X + \sigma_0 \text{ [L2R]}$$

Implementation of the module computing error locator polynomial from right to left:
The first architecture of the module computing error locator polynomial from right to left [R2L] is presented in Figure 1 and the implementation results are presented in Table 2. As it can be seen,

this version needs two GMs and two registers for saving intermediate results. Coefficients σ_i and error vector are saved in separate embedded memory. The maximum frequency of this block is much lower than that of GM alone, because of memory accesses (coefficients σ_i are read from the memory on the fly).

Figure 1: Architecture for [R2L] evaluation

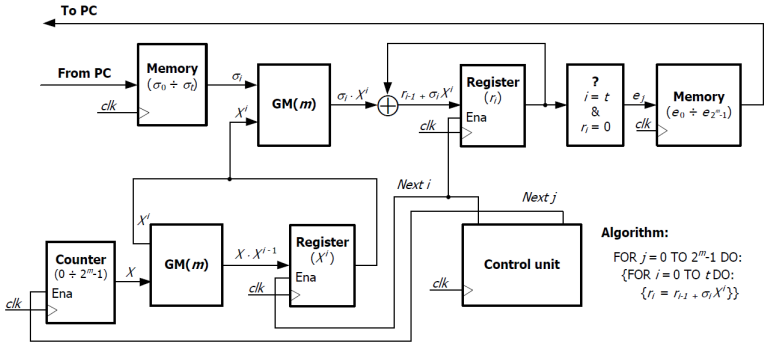


Table 2: Implementation results for [R2L] evaluation

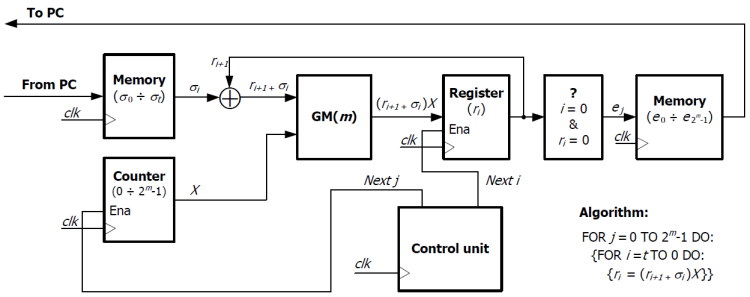
m	t	LCELLs	freq (MHz)	time (μs)
5	3	287	120	1.07
6	5	311	118	3.25
7	10	341	95	14.82
11	50	511	78	1339.08

Implementation of the module computing error locator polynomial from left to right: The second architecture of the module computing error locator polynomial from left to right [L2R] according to Horner's scheme is depicted in Figure 2 and the implementation results are presented in Table 3. As it can be seen, this architecture needs only one GM and one register. Memories for coefficients σ_i and error vector (the result) are the same as in the previous case.
Of course, both architecture versions give the same error vector as a result. However, intermediate results, which are saved in registers, are different. Therefore, both circuits will leak information differently. We expect that this difference can be used for constructing some efficient countermeasure in the future.

Table 3: Implementation results for [L2R] evaluation

m	t	LCELLs	freq (MHz)	time (μs)
5	3	274	110	1.16
6	5	293	110	3.49
7	10	311	107	13.16
11	50	414	85	1228.80

Figure 2: Architecture for [R2L] evaluation



5 Conclusions and perspectives

Goppa codes are one of the most used family of codes in code-based cryptography. We implemented a part of the algorithm, which is the most expensive, vulnerable but necessary in all Goppa decoding algorithms. Implementation results of both versions ([R2L] and [L2R]) are very similar. However, power traces will be certainly very different. Moreover, we can ask us if it will be possible to attack both implementations. If not, which is more robust and the reasons, else we can implement both methods in parallel and select randomly the datapath. It will be necessary to determine which is the best Goppa decoder between Patterson, Berlekamp-Massey or Extended Euclidean algorithms. Later, we could implement the complete Goppa decoder and the complete McEliece PKC in hardware in order to evaluate side-channel attacks and countermeasures.

References

[1] Evariste II - A Modular Hardware System for Development and Evaluation of Cryptographic Functions and Random Number Generators. `http://labh-curien.univ-st-etienne.fr/wiki-evariste-ii/`.

[2] R.M. Avanzi, S. Hoerder, D. Page, and M. Tunstall. Side-Channel Attacks on the McEliece and Niederreiter Public-Key Cryptosystems. 2010.

[3] E. Berlekamp. Goppa codes. *Information Theory, IEEE Transactions on*, 19(5):590–592, 1973.

[4] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384 – 386, may 1978.

[5] N.T. Courtois, M. Finiasz, and N. Sendrier. How to Achieve a McEliece-Based Digital Signature Scheme. 2248:157–174, 2001.

[6] V. D. Goppa. A new class of linear correcting codes. *Problemy Peredachi Informatsii*, 6(3):24–30, 1970.

[7] S. Heyse and T. Güneysu. Code-based cryptography on reconfigurable hardware: tweaking niederreiter encryption for performance. *Journal of Cryptographic Engineering*, pages 1–15, 2013.

[8] W. G. Horner. A new method of solving numerical equations of all orders, by continuous approximation. *Philosophical Transactions of the Royal Society of London*, 109:308–335, 1819.

[9] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology (CRYPTO'96)*, pages 104–113. Springer, 1996.

[10] J. Massey. Shift-register synthesis and BCH decoding. *Information Theory, IEEE Transactions on*, 15(1):122–127, 1969.

[11] R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, 42(44):114–116, 1978.

[12] H. Molter, M. Stöttinger, A. Shoufan, and F. Strenzke. A simple power analysis attack on a McEliece cryptoprocessor. *Journal of Cryptographic Engineering*, 1(1):29–36, April 2011.

[13] N. Patterson. The Algebraic Decoding of Goppa Codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.

[14] A. Shoufan, F. Strenzke, H. Molter, and M. Stöttinger. A Timing Attack against Patterson Algorithm in the McEliece PKC. *Information, Security and Cryptology–ICISC 2009*, 5984:161–175, 2010.

[15] F. Strenzke. A Timing Attack against the secret Permutation in the McEliece PKC. *Post-Quantum Cryptography*, 6061:95–107, 2010.

[16] F. Strenzke. Timing Attacks against the Syndrome Inversion in Code-based Cryptosystems. 2011.

[17] F. Strenzke, E. Tews, H. G. Molter, R. Overbeck, and A. Shoufan. Side Channels in the McEliece PKC. *Post-Quantum Cryptography*, 5299(5299/2008):216–229, October 2008.