Towards a secure implementation of a Goppa decoder (Work in progress)

Tania RICHMOND
with Pierre-Louis CAYREL, Viktor FISCHER
and Pascal VÉRON

June 25th, 2013

Error-correcting codes

Original application: Correct errors after data transmission (if possible).
How? Adding some redundant information.

Example

Messages: $\{0, 1\}$ Codewords: $\{000, 111\}$
$1 \rightarrow 111 \longrightarrow 110 \rightarrow 111 \rightarrow 1$

We use linear codes so the redundancy is linearly dependant of the information.

Code-based cryptography

New application of error-correcting codes:

- Fast computations,
- No quantum algorithm with polynomial complexity to break the mathematical hard problem (to date).

Principle: uses syndrome decoding as a trapdoor one-way function

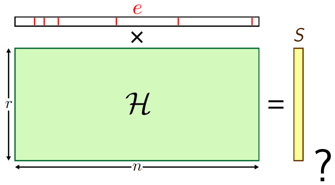
Syndrome decoding problem:

Known:

$\mathcal{H}$ a $r \times n$-matrix;
S a vector of length r;
and t an integer $< n$.

Question:

Does there exist a vector e of length n and weight t such that:



Goppa codes used as trapdoor

Goppa decoder can be used both for encryption and signature.

Used in

- The McEliece public-key encryption scheme;
- The CFS signature scheme.

Advantages of Goppa codes:

- Like random codes in several (most) cases,
- Dense family of codes,
- Have an efficient decoding algorithm.

Patterson algorithm

- Used in code-based cryptography;
- For binary Goppa codes decoding;
- Proposed by N. J. Patterson in 1975.

Advantages

Fast computations for binary codes and ability to correct more errors.

Patterson algorithm
McEliece cryptosystem

McEliece Public-Key Cryptosystem:

KeyGen

Public key: Choose a code, with its generator matrix, for which we have a decoding algorithm.

Private key: Transform this generator matrix to obtain an equivalent generator matrix which seems random.

Encrypt

Encode the message and add a random error of weight t.

Decrypt

Decode the ciphertext and come back to the original codeword.

Patterson algorithm
Input/output

Inputs:

$y = c \oplus e$ a codeword with errors,
$\Gamma(L, g(X))$ the Goppa code, with
$L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subset \mathbb{F}_{2^m}$ and
g a polynomial of degree t.

Output:

c a codeword (without errors).

All operations are in finite fields.

Patterson algorithm

Algorithm overview

1. Compute the syndrome polynomial.
2. Invert the syndrome polynomial modulo $g(X)$.
3. Compute the square root of the syndrome polynomial inverse plus X modulo $g(X)$.
4. Determine the even and odd parts of the error locator polynomial (ELP).
5. Determine the ELP.
6. Evaluate the ELP to find its roots.
7. Correct the codeword with the error vector.

Patterson algorithm
Step 0 (precomputation)

Square root of $X \mod g(X)$

$$\begin{aligned} R(X) &= \sqrt{X} && \mod g(X) \\ &= X^{2^{m-1}} && \mod g(X) \end{aligned}$$

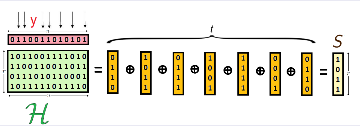
because for all $X \in \mathbb{F}_{2^m}$, X verifies $X^{2^m} = X$, so $X^{2^{m-1}} = X^{1/2}$.

Patterson algorithm

Step 1

Syndrome polynomial

Compute $S = \mathcal{H}^t y$ summing several columns of $\mathcal{H}$.



Then we obtain the Syndrome polynomial $S(X)$.

Example (from S to $S(X)$)

We can see the vector $S = (s_{t-1}, s_{t-2}, \ldots, s_1, s_0)$
as the polynomial $S(X) = s_{t-1}X^{t-1} + s_{t-2}X^{t-2} + \ldots + s_1 X + s_0$.

Patterson algorithm
Step 2

Syndrome inversion

Compute

$$T(X) = S^{-1}(X) \mod g(X)$$

by extended Euclidean algorithm.

Patterson algorithm

Step 3

Square root of Syndrome inverse plus X, modulo $g(X)$

Compute

$$\tau(X) = \sqrt{T(X) + X} \mod g(X).$$

Setting $h(X) = T(X) + X$, we use the following formula:

$$\tau(X) = \sum_{i=0}^{(t-1)/2} h_{2i}^{2^{m-1}} X^i + \sum_{i=0}^{t/2-1} h_{2i+1}^{2^{m-1}} X^i R(X)$$

where $R(X) = \sqrt{X} \mod g(X)$.

Patterson algorithm
Step 4

Even and odd parts of the Error Locator Polynomial (ELP)

Compute $a(X)$ and $b(X)$ such that

$$a(X) = b(X)\tau(X) \mod g(X)$$

by extended Euclidean algorithm.

Patterson algorithm
Step 5

ELP building

Construct $\sigma(X)$ such that:

$$\sigma(X) = a^2(X) + Xb^2(X)$$

Example (if t is odd)

$$a(X) = \sigma_0 + \sigma_2 X + \ldots + \sigma_{t-1} X^{(t-1)/2}$$
$$b(X) = \sigma_1 + \sigma_3 X + \ldots + \sigma_t X^{\lfloor t/2 \rfloor}$$
$$\sigma(X) = \sigma_0 + \sigma_1 X + \ldots + \sigma_{t-1} X^{t-1} + \sigma_t X^t$$

Patterson algorithm
Step 6

Finding roots of the ELP

Test all elements in L to find the roots of σ.
The roots of σ correspond to the nonzero coordinates of the error vector.

Example (Horner algorithm)

$$\begin{aligned}\sigma(X) &= \sigma_t X^t + \sigma_{t-1} X^{t-1} + \ldots + \sigma_1 X + \sigma_0 \\ &= (((\sigma_t X + \sigma_{t-1})X + \ldots)X + \sigma_1)X + \sigma_0\end{aligned}$$

for all X in L.

Patterson algorithm

Step 7

Error vector reconstruction

Change all corresponding coordinates in y.

(Thanks to the error vector found in the previous step.)

Implementation in hardware

Core unit - Galois field multiplier (GM) (1/4)

Steps 5 and 6 of Patterson algorithm in hardware:

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 1)

Inputs: Two elements of $\mathbb{F}_{2^m}$ represented by two polynomials of degree $m-1$

$$\alpha(X) = \sum_{i=0}^{m-1} \alpha_i X^i \text{ and } \beta(X) = \sum_{i=0}^{m-1} \beta_i X^i$$

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 2)

Output: Product of the inputs seen as a polynomial of degree $m-1$

$$r(X) = \sum_{i=0}^{m-1} r_i X^i$$

Implementation in hardware

Core unit - Galois field multiplier (2/4)

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
$\beta(x) = x^4 + x$

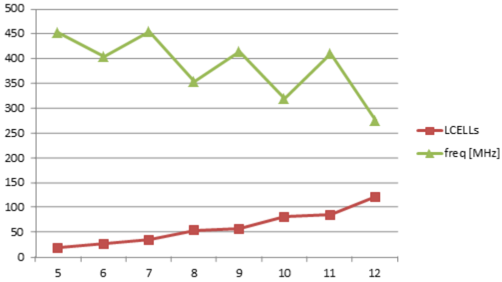
	x^5	x^4	x^3	x^2	x	1
Q_m	1	0	1	1	1	1
α		1	0	1	1	0
β		1	0	0	1	0
r		1	0	0	1	0
$(i = 4)$	1	0	0	1	0	0
		0	0	0	0	0
	1	0	1	1	1	1
r	0	0	1	0	1	1
$(i = 3)$			$\vdots$			

Implementation in hardware
Core unit - Galois field multiplier (3/4)

Version 1:

- Implementation results
- Hardware: Altera Cyclone III FPGA (EP3C25)
- With fixed Q_m

m	LCELLs	freq [MHz]
5	21	453
6	29	405
7	37	455
8	56	355
9	59	414
10	83	321
11	87	411
12	123	276



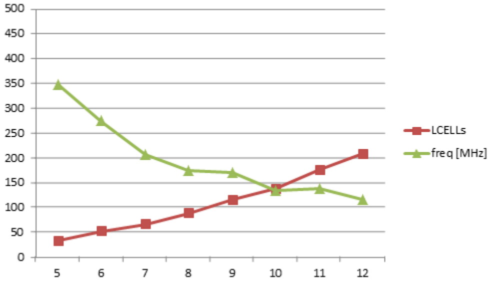
Implementation in hardware

Core unit - Galois field multiplier (4/4)

Version 2:

- Implementation results
- Hardware: Altera Cyclone III FPGA (EP3C25)
- With variable Q_m

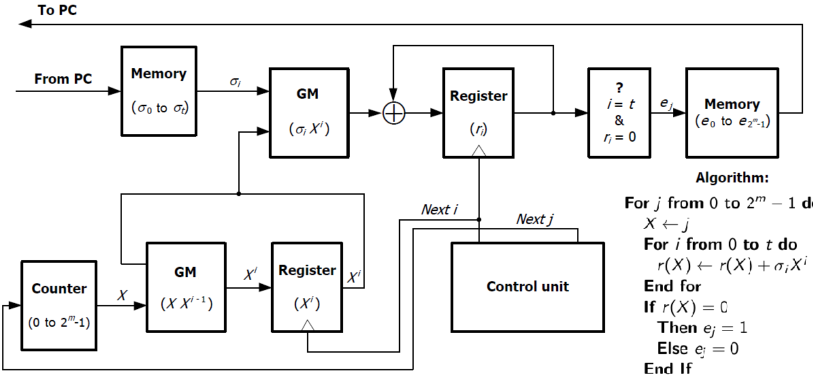
m	LCELLs	freq [MHz]
5	35	348
6	54	275
7	68	207
8	90	175
9	117	172
10	140	135
11	178	140
12	209	118



Implementation in hardware

Architecture of error vector computation (Right to Left [R2L])

Expression: $\sigma(X) = \sigma_t X^t + \sigma_{t-1} X^{t-1} + \ldots + \sigma_1 X + \sigma_0$



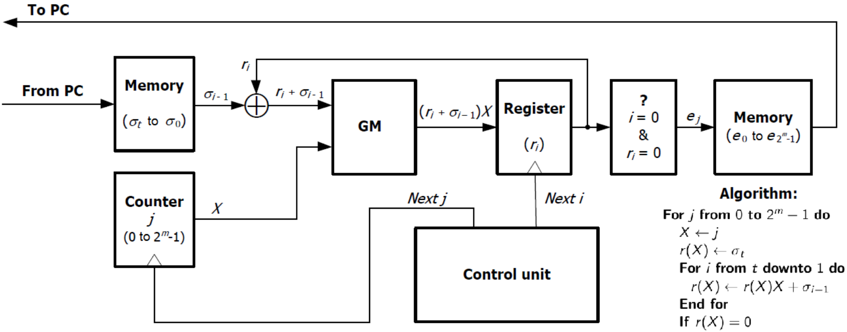
Algorithm:

For j from 0 to $2^m - 1$ **do**
  $X \leftarrow j$
  For i from 0 **to** t **do**
    $r(X) \leftarrow r(X) + \sigma_i X^i$
  End for
  If $r(X) = 0$
    Then $e_j = 1$
    Else $e_j = 0$
  End If
End for

Implementation in hardware

Architecture of error vector computation (Left to Right [L2R])

Expression: $\sigma(X) = (((\sigma_t X + \sigma_{t-1})X + \ldots)X + \sigma_1)X + \sigma_0$



Algorithm:

For j **from** 0 **to** $2^m - 1$ **do**
  $X \leftarrow j$
  $r(X) \leftarrow \sigma_t$
  For i **from** t **downto** 1 **do**
    $r(X) \leftarrow r(X)X + \sigma_{i-1}$
  End for
  If $r(X) = 0$
    Then $e_j = 1$
    Else $e_j = 0$
  End If
End for

Implementation in hardware

Error vector computation - implementation results

- Hardware: Altera Cyclone III FPGA (EP3C25)
- With fixed Q_m (version 1)
- Implementation results [R2L]

m	t	LCELLs	M9K	freq [MHz]	Total[µs]
6	5	296	1 + 2	135	2.84
7	10	317	1 + 2	130	10.83
11	50	447	1 + 2	123	849.17

- Implementation results [L2R]

m	t	LCELLs	M9K	freq [MHz]	Total[µs]
6	5	279	1 + 2	125	3.07
7	10	295	1 + 2	123	11.48
11	50	378	1 + 2	117	892.72

Conclusion

- Goppa codes are one of the most used family (of codes) in code-based cryptography.
- Implemented a part of the algorithm, which is the most expensive, vulnerable and 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.

Go to a secure implementation of a Goppa decoder.

Future works

- Will it be possible to attack both implementations?

  No: Which is more robust and why?

  Yes: We can implement both methods in parallel and select randomly the datapath.
- Determine which is the best Goppa decoder between:
  1. Patterson algorithm,
  2. Berlekamp-Massey algorithm,
  3. Extended Euclidean algorithm.
- Implementation of the complete Goppa decoder in hardware.
- Implementation of the complete McEliece cryptosystem in hardware.
- Evaluations of side-channel attacks and countermeasures.

Towards a secure implementation of a Goppa decoder (Work in progress)

Thank you for your attention.

Questions ?