Don't forget your roots: constant-time root finding over $\mathbb{F}_{2^m}$

Douglas Martins[1] Gustavo Banegas[2,3]
Ricardo Custódio[1]

[1]Departamento de Informática e Estatística,
Universidade Federal de Santa Catarina

[2]Department of Mathematics and Computer Science
Technische Universiteit Eindhoven

[3]Department of Computer Science and Engineering
Chalmers Tekniska Högskola

October 2
LATINCRYPT 2019

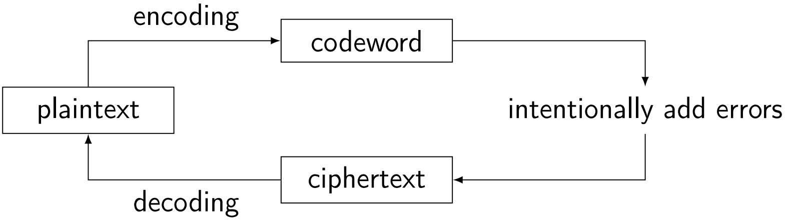
Outline

Introduction

- Traditional algorithms used in cryptography are insecure against a quantum adversary
  - Post-quantum cryptography algorithms aim to provide security in a quantum era
- NIST standardization process is looking for new algorithms, and one of the targets are Key Encapsulation Mechanisms (KEMs)
  - Cryptosystems based on coding theory are candidates to create safe KEMs

McEliece Cryptosystem

- Robert J. McEliece proposed the first cryptosystem based on coding theory [McE78]
  - Until today, most code-based cryptosystems are based on the same structure



McEliece Cryptosystem

Key generation and encryption process

- Given a Goppa code $\Gamma(L, g(z))$, where $g(z) \in \mathbb{F}_{2^m}$ is the Goppa polynomial and $L = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ the support, then we can generate a key pair for a McEliece instance as:
  - *Public key:* $pk = G$, such that G is a generator matrix from Γ
  - *Secret key:* $sk = (L, g(z))$
- Given a message $m \in \mathbb{F}_2^k$, we encrypt this message by encoding m using the generator matrix G, then we XOR it with a random error vector e with length n and Hamming weight t
  - *Encryption process:* $c = m \times G \oplus e$

McEliece Cryptosystem

Decoding process

- The decoding process was made efficient through Patterson's algorithm [Pat75]
  - Other decoders could be used for this task, although some of them require larger key sizes
- The main idea of Patterson's algorithm is to compute the syndrome value $S_c(z)$ from a received word c, after that, it defines the **error locator polynomial (ELP)**, or $\sigma(x)$, for c
- The positions of the roots of σ in L define the position where an error was added

Side-channel attacks

- As shown by [SSMS09] and [BCDR17], timing side-channel attacks could be done during the computation and factorization of ELP
  - A naive implementation for the factorization of ELP enables an attacker to recover the plain text
- In [Str12] demonstrates algorithms to find roots efficiently in code-based cryptosystems
  - However, the author shows only timings in different types of implementations and selects the one that has the least timing variability
- [BCS13] uses Fast Fourier Transform to achieve a secure decoding, but is built and optimized for $\mathbb{F}_2^{13}$

Attack on BIGQUAKE

BInary Goppa QUAsi-cyclic Key Encapsulation

- BIGQUAKE is a round 1 submission to NIST standardization process that uses binary Quasi-cyclic (QC) Goppa codes in order to accomplish a KEM between two distinct parties
- The main idea of the algorithm was based on a message encrypted with a public key. After that, the receiver decodes the ciphertext, removing the error added to the message

Attack on BIGQUAKE

BInary Goppa QUAsi-cyclic Key Encapsulation

- As argued, a naive implementation of the decoding step is vulnerable to side-channel attacks and we use this fact to perform the attack presented in [SSMS09]
- The attack exploits the fact that flipping a bit of the error e changes the Hamming weight and per consequence, the timing for decryption
- Using a precision parameter $M = 500$, it took ≈ 17 minutes to recover a message m

Root finding methods

- We are interested in constructing a way to compute the roots of σ without leaking information of which error was added to the original message
- We present four countermeasures for root finding methods which are used in code-based cryptosystems
  - Exhaustive search
  - Linearized polynomials
  - Berlekamp Trace Algorithm
  - Successive Resultant Algorithm

Exhaustive search

- The exhaustive search is a direct method which makes a sequential evaluation of all possible values in σ
  - Saving one element in a list when a root is found implies in a extra operation that could be detected in a side-channel attack
- Our main countermeasure is to permute all elements before evaluating the root candidate
- Using this technique, an attacker can identify the extra operation, but cannot learn any secret information
  - In our proposal, we employ the Fisher-Yates shuffle

Linearized polynomials

- The second countermeasure proposed is based on the computation of roots over a class of polynomials called linearized polynomials
  - In [FT02], the authors propose a method for root finding over a polynomial as $\ell(y) = \sum_i c_i y^{2^i}$
- In addition, from [TJR01], we have the definition of an affine polynomial
  - $A(y)$ over $\mathbb{F}_{2^m}$ is an affine polynomial if $A(y) = \ell(y) + \beta$ for $\beta \in \mathbb{F}_{2^m}$, where $\ell(y)$ is a linearized polynomial

Linearized polynomials

- In [FT02], the authors provide a generic decomposition for finding affine polynomials

$$f(y) = f_3 y^3 + \sum_{i=0}^{\lceil (t-4)/5 \rceil} y^{5i} (f_{5i} + \sum_{j=0}^{3} f_{5i+2^j} y^{2^j})$$

  - We use Gray codes for the generation of the elements in $\mathbb{F}_{2^m}$ to find the roots of σ
- We add countermeasures in the algorithm in order to blind the branches, adding a operation with the same cost for each branch

Berlekamp Trace Algorithm

- Given a trace function $Tr(x) = \sum_{i=0}^{m-1} x^{2^i}$ and a standard basis $\beta = \{\beta_1, \dots \beta_m\}$, the BTA is described as:

Algorithm 1: $BTA(p(x), i)$ (recursive version)

if $deg(p(x)) \leq 1$ **then**
    return root of $p(x)$
end
$p_0(x) \leftarrow gcd(p(x), Tr(\beta_i \cdot x))$
$p_1(x) \leftarrow QuoRem(p(x), p_0(x))$
return $BTA(p_0(x), i+1) \cup BTA(p_1(x), i+1)$

- The recursive behavior of BTA is the main drawback against a side-channel attack
  - Additionally, trace functions can reach non-divisors of the current polynomial, making some iterations worthless

Berlekamp Trace Algorithm

- To avoid this time variance, we propose a new iterative version of BTA

Algorithm 2: *BTA*$(p(x))$ (iterative version)

$g \leftarrow \{p(x)\}$ `// polynomials to be computed`
for $k \leftarrow 0$ **to** t **do**
  $current = g.pop()$
  Compute $candidates = gcd(current, Tr(\beta_i \cdot x)) \; \forall \; \beta_i \in \beta$
  Select $p_0 \in candidates$ such that $p_0.degree \simeq \frac{current}{2}$
  $p_1(x) \leftarrow QuoRem(current, p_0(x))$
  if $p_0.degree == 1$ **then** $R.add($root of $p_0)$
  else $g.add(p_0)$
  if $p_1.degree == 1$ **then** $R.add($root of $p_1)$
  else $g.add(p_1)$
end
return R

Successive Resultant Algorithm

- Proposed in [Pet14] and generalized in [DPP16], the SRA relies on the fact that it is possible to find roots exploiting properties of an ordered set of rational mappings
- The main idea of the algorithm is to construct a polynomial system such that

$$\begin{cases} f(x_1) = 0 \\ x_j^p - a_j x_j = x_{j+1}, \quad j = 1, \ldots, n-1 \\ x_n^p - a_n x_n = 0 \end{cases} \tag{1}$$

Successive Resultant Algorithm

- From [Pet14], if $(x_1, x_2, \ldots, x_m)$ is a solution for Equation 1, then $x_1 \in \mathbb{F}_{p^m}$ is a root of f
  - Conversely, given a solution $x_1 \in \mathbb{F}_{p^m}$ of f, we can reconstruct a solution of all equations in Equation 1 by setting $x_2 = x_1^p - a_1 x_1$ etc.
  - In [Pet14], the authors present an algorithm for solving the system in Equation 1 using resultants
- It is worth remarking that this algorithm is almost constant-time and hence we just need to protect the branches presented on it

Results

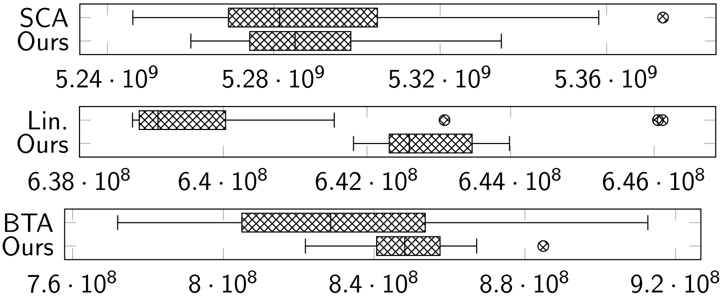


Figure: Comparison of CPU cycles of original implementation and our proposal for Linearized, Successive resultant algorithm and Berlekamp trace algorithm with $t = 100$.

Open problems

- Improve our implementation using vectorization, bit slicing or Intel® IPP Cryptography instructions for finite fields
- Improve security analysis by removing conditional memory access
  - Consider different attack scenarios and perform an analysis of hardware side-channel attacks
- Analysis of different methods to compute roots, and check their security against side-channel attacks

Thank you for the attention!

marcelino.douglas@posgrad.ufsc.br

References I

Dominic Bucerzan, Pierre-Louis Cayrel, Vlad Drağoi, and Tania Richmond.
Improved timing attacks against the secret permutation in the McEliece PKC.
International Journal of Computers Communications & Control, 12(1):7–25, 2017.

Daniel J Bernstein, Tung Chou, and Peter Schwabe.
McBits: fast constant-time code-based cryptography.
In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 250–272. Springer, 2013.

James H. Davenport, Christophe Petit, and Benjamin Pring.
A Generalised Successive Resultants Algorithm.
In Sylvain Duquesne and Svetla Petkova-Nikova, editors, *Arithmetic of Finite Fields*, pages 105–124, Cham, 2016. Springer International Publishing.

Sergei V Fedorenko and Peter V Trifonov.
Finding roots of polynomials over finite fields.
IEEE Transactions on communications, 50(11):1709–1711, 2002.

Robert J McEliece.
A Public-Key Cryptosystem Based On Algebraic Coding Theory.
Deep Space Network Progress Report, 44:114–116, January 1978.

References II

Nicholas Patterson.
The algebraic decoding of Goppa codes.
IEEE Transactions on Information Theory, 21(2):203–207, 1975.

Christophe Petit.
Finding roots in GF(p^n) with the successive resultant algorithm.
IACR Cryptology ePrint Archive, 2014:506, 2014.

Abdulhadi Shoufan, Falko Strenzke, H. Gregor Molter, and Marc Stöttinger.
A timing attack against patterson algorithm in the McEliece PKC.
In *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, pages 161–175, 2009.

Falko Strenzke.
Fast and secure root finding for code-based cryptosystems.
In *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, pages 232–246, 2012.

T-K Truong, J-H Jeng, and Irving S Reed.
Fast algorithm for computing the roots of error locator polynomials up to degree 11 in Reed-Solomon decoders.
IEEE Transactions on Communications, 49(5):779–783, 2001.