# Efficient Root Finding of Polynomials over Fields of Characteristic 2.

**Bhaskar Biswas & Vincent Herbert**

CRI INRIA Paris-Rocquencourt
Domaine de Voluceau - Le Chesnay - 78153 - B.P. 105 - France
Bhaskar.Biswas@inria.fr; Vincent.Herbert@inria.fr

**Abstract.** Root finding is the most time-consuming stage of McEliece cryptosystem decryption. The best method to find the zeroes of a polynomial for cryptographic parameters is the Berlekamp Trace Algorithm (BTA). Our idea is to mix BTA with ad-hoc methods proposed by Zinoviev. We obtain a significant gain in terms of time complexity for finding roots and so we decrease McEliece decryption time. This paper contains both theoretical and experimental study of this technique.

**Keywords:** Worst-case complexity, Code-based public-key cryptography, Linearized polynomial, Berlekamp Trace Algorithm, Error locator polynomial, Algebraic decoding algorithm, Binary Goppa codes.

## 1 Introduction

Root finding of polynomials over finite fields is a classical algebraic algorithmic problem. It is considered as one of the most time-consuming subprocess of the decoding process of Reed-Solomon, BCH and Goppa codes. There are some well known approaches for finding roots of the so-called error locator polynomial. The most widely known root finding algorithm is Chien search method [Chi64], which is an evaluation of the polynomial at all elements of the field, so it has very high time complexity for the case of large fields and polynomials of high degree. Berlekamp Trace Algorithm (BTA) [Ber71] is another well known method. It is a recursive method based on the trace function properties.

McEliece cryptosystem is considered as one of the fastest public key schemes and is still esteemed secure for reasonable parameters. The classical [McE78] (described in Appendix A) and hybrid [BS08] McEliece schemes use binary Goppa codes [Gop70]. The decryption process employs an algebraic decoding algorithm which is often broken up in three parts namely: syndrome computation, finding the solution of the key equation,

and the root finding of the error locator polynomial. This last step takes theoretically three fourth of the total decryption time.

In this paper, we present a hybrid method involving BTA and a method proposed by Zinoviev [Zin96]. Zinoviev proposed direct root finding procedures for polynomials with degree at most 10. Our idea is to compute directly the roots with Zinoviev procedures up to some degree and to use BTA for greater degrees. Moreover, we improve Zinoviev procedures for polynomials of degree 2 and 3 with time-memory tradeoffs. We analyze both the theoretical complexities and the experimental complexities of our proposal. We obtain a theoretical gain of 93% over Chien method and 46% over BTA. Experimental results confirm theory up to degree 4 at least. For instance with $m = 11$, $t = 32$ and $d_{max} = 4$, our method takes 60% of the total decryption time with respect to 72% for BTA and 87% for Chien.

The paper is structured as follows. In Section 2, we explain our motivation for root finding of polynomials over binary fields. Section 3 is dedicated to related works. In Section 4, we state our proposal and in Section 5, we present the simulation results to back up our proposal.

## 2 Motivation

Let $\mathbb{F}_{2^m}$ be the extension field of degree $m$ of the two-element field $\mathbb{F}_2$. We consider a univariate monic polynomial $f$, of degree $t > 0$, in the polynomial ring $\mathbb{F}_{2^m}[z]$. Without loss of generality, we assume that $f$ has no multiple root and that $f$ factorizes into linear factors over the binary field $\mathbb{F}_{2^m}$ (*e.g.* see [LN96]). Our goal is to find an efficient way, in terms of time and space complexity, to find all the zeroes of $f$.

We are specifically interested in the said problem in the McEliece context.

The efficiency of the root finding algorithms is a problem that we study in code-based cryptography. McEliece-type cryptosystems are often based on binary Goppa codes (presented in Appendix B). Their decryption algorithm employs an algebraic decoding process to recover the original message from the cyphertext. The most time-consuming stage, in the implementation of algebraic decoding of binary Goppa codes, with practical parameters, is the root finding of the error locating polynomial. This polynomial fulfills the above mentioned properties.

In this article, we consider an $n$-lenght binary Goppa code that corrects up to $t$ errors (the algorithm used is given in Appendix B). Let us recall, in practice, parameters are chosen such that: $n = 2^m$ and $mt \leq n$.

**Decryption Complexity**

Theoretical Complexity = number of arithmetic operations in $\mathbb{F}_{2^m}$ required to decrypt in the worst case.

- Syndrome computation $\mathcal{O}(nt)$
- Key equation solving $\mathcal{O}(t^2)$
- Error locator polynomial root finding
  - BTA $\mathcal{O}(mt^2)$
  - Chien search $\mathcal{O}(nt)$

Experimental Complexity = average running time for the decryption.

We give below the percentage of the total time spent in each stage of the decryption algorithm.

- Syndrome computation 11.3%
- Key equation solving 12.0%
- Error locator polynomial root finding
  - BTA 72.1%
  - Chien search 85.6% [1]
- Other tasks 4.6%

Asymptotically, syndrome computation is the leading cost. For recommended parameters (*i.e.* $m = 11$, $t = 32$), the most time-consuming step in the decryption (decoding) consists in finding roots of $\sigma_e$. Differences that can appear between theoretical and experimental complexities would be due to several reasons:

- tweaks of implementation (quality of implementation, used processors and compilers);
- cost of conditional tests and memory accesses (that are neglected in our theoretical study but that can be noteworthy in practice);
- not so good approximations (*e.g.* we approximate the cost of an inversion of a binary matrix of order $m$ to $m^2$ field operations, we thus overlook a small multiplicative constant);

[1] The percentages given are associated with BTA, if we take Chien search, the numbers for syndrome computation and key equation solving should vary accordingly.

– weighting[2] of arithmetic operations in the field.

## 3 Related Works

Several approaches toward root finding in characteristic 2 are possible, their efficiency depends on the size of the parameters $m$ and $t$.

– Chien search (described in Appendix C) computes roots by evaluating artfully the polynomial in all points of $L$. This method is recommended for hardware implementations and coding theory applications in which $m$ is small.
– BTA is a recursive algorithm using trace function properties. It is a faster method for secure parameters for McEliece-type cryptosystems.
– Equal-degree factorization is an algorithm of Cantor and Zassenhaus [vzGG03, Chapter 14]. Its scope is more general but under some adaptations, it enables to find roots of polynomials in characteristic 2 (this specific case is treated in Exercise 14.16 in [vzGG03]). Then it has similarities with BTA (use of trace function, computations of gcds and a recursive structure) but seems slightly more expensive. Its time complexity is $\mathcal{O}((m + \log t)\ t^2 \log t)$ operations in $\mathbb{F}_{2^m}$.
– Zinoviev procedures are dedicated to root finding for polynomials of degree less than 10. For $m \geq 11$, they are (theoretically at least) more efficient than Chien search.

We present below the overview of BTA and Zinoviev procedures that we employ to consummate our proposal.

### 3.1 Berlekamp Trace Algorithm (1971)

We define the *Trace function* as $\mathrm{Tr}(\cdot) : \mathbb{F}_{2^m} \to \mathbb{F}_2$

$$\mathrm{Tr}(z) := z + z^2 + z^{2^2} + \ldots + z^{2^{m-1}}.$$

$\mathrm{Tr}(\cdot)$ is a $\mathbb{F}_2$-linear mapping and we know that: $z^{2^m} - z = \mathrm{Tr}(z) \cdot (\mathrm{Tr}(z) - 1)$.

The *Trace function* has the following property:

$$\mathrm{Tr}(z) - i = \prod_{\gamma \ s.t.\ \mathrm{Tr}(\gamma) = i} (z - \gamma),\ \forall i \in \mathbb{F}_2.$$

[2] In our study, we distinguish two cases: in the first one, addition and multiplication both cost one operation in $\mathbb{F}_{2^m}$, we denote it $K(+) = K(\times) = 1$ where $K$ is the cost function of an arithmetic operation, in the second one, $K(+) = 1$ and $K(\times) = m$.

Let $B = (\beta_1, \ldots, \beta_m)$ be a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Every element $\alpha \in \mathbb{F}_{2^m}$ is uniquely represented by the binary m-tuple $(\mathrm{Tr}(\beta_1 \cdot \alpha), \ldots, \mathrm{Tr}(\beta_m \cdot \alpha))$.

BTA splits any $f \in \mathbb{F}_{2^m}[z]$, such that $f(z)|(z^{2^m} - z)$, into linear factors by computing iteratively on $\beta \in B$ and recursively on $f$:

$$g(z) := \gcd(f(z), \mathrm{Tr}(\beta \cdot z)) \text{ and } h(z) := \frac{f(z)}{g(z)} = \gcd(f(z), \mathrm{Tr}(\beta \cdot z) - 1).$$

First call: $f = \sigma_e$ and $\beta = \beta_1$. BTA always successfully returns the linear factors of $f$ due to the properties of the trace given above. More details on BTA can be found in [Ber71] and [MvOV88]. In practice, we precompute the polynomials $\mathrm{Tr}(\beta_i \cdot z) \bmod f(z)$, $\forall i \in \{0, \ldots, m-1\}$. The cost of this precomputation is $\mathcal{O}(mt^2)$. We stress on the fact that it is not a negligible cost. Indeed, let us recall that BTA without precomputation has also a cost of $\mathcal{O}(mt^2)$ operations over $\mathbb{F}_{2^m}$.

### 3.2 Zinoviev Procedures (1996)

Zinoviev methods [Zin96] find the monic affine multiple of smallest degree of any polynomial $f$ of degree $d \leq 10$ over $\mathbb{F}_{2^m}$. At step $i \geq 0$, we compute a multiple of $f$ of degree $2^{\lceil \log_2 d \rceil + i}$ and we try to decimate the non-linear terms by solving a homogeneous system of linear equations. If the system has no solution, we go up step $i + 1$. Besides, an algorithm proposed by Berlekamp, Rumsey and Solomon in [BRS67] ensures to find an affine multiple of degree $2^{d-1}$ and thus guarantee Zinoviev methods terminate, in the worst case, at step $d - 1 - \lceil \log_2 d \rceil$. After that, finding roots of an affine polynomial is easier than in the general case (see Appendix D). For this, we only have to solve a linear system of order $m$ over $\mathbb{F}_2$. Then, we have to determine the roots of $f$, among the roots of the affine polynomial we have found. We just evaluate $f$ in those points to do so.

Consider $q$ is a prime power and $m$ is a positive integer. Let us give the useful definitions:

**Definition 1.** *An affine polynomial has the form:*

$$A(z) = L(z) + c$$

*where $L$ is a linearized polynomial over $\mathbb{F}_{q^m}$ and $c \in \mathbb{F}_{q^m}$.*

**Definition 2.** *A linearized polynomial over* $\mathbb{F}_{q^m}$ *is a polynomial of the form:*

$$L(z) = \sum_{i=0}^{n} l_i \cdot z^{q^i}.$$

*with* $l_i \in \mathbb{F}_{q^m}$ *and* $l_n = 1$.

In our case, $q = 2$. The *Trace polynomial* is an example of linearized polynomial.

## 4 Our Proposal

### 4.1 Speed up McEliece Decryption

The drawback of BTA is the large number of recursive calls when the system parameters grow. We reduce it by mixing BTA and Zinoviev procedures that are ad-hoc methods for finding roots of polynomials of degree $\leq 10$. This is a classical technique employed to decrease the number of recursive levels, *e.g.* the well-known Quicksort algorithm is optimized with analagous methods. We call this process BTZ (Berlekamp Trace - Zinoviev) in the scope of this document. BTZ depends on a parameter $d_{max}$ that is the maximum degree up to which we use Zinoviev methods. We give two pseudocodes of BTZ in Appendix F.

### 4.2 Implementation Tweaks

In our implementation, we use a polynomial basis to represent the field elements. We implemented Zinoviev procedures with time-memory tradeoffs for polynomials of degree 2 and 3, that enable to perform better than with the original procedures. We explain these new methods and give their complexities in the following.

**Time-Memory Tradeoff for Degree 2.** We want to solve the equation: $z^2 + az + b = 0$ for $a$, $b \in \mathbb{F}_{2^m}$. If the solutions exist, we denote them $z_1$ and $z_2$. First, we make a change of variable. We set $z = ax$. We obtain the equation $x^2 + x + b/a^2 = 0$. It costs one division and one squaring in $\mathbb{F}_{2^m}$.

Let $i$ be an element of $\mathbb{F}_{2^m}$ and $f_i$ be the mapping:

$$\begin{aligned} f_i : \ \mathbb{F}_{2^m} &\to \mathbb{F}_{2^m} \\ x &\mapsto x^2 + x + i \end{aligned}$$

The equation $f_i(x) = 0$ has two solutions in $\mathbb{F}_{2^m}$ if and only if $\mathrm{Tr}(i) = 0$ (a proof is given in [BRS67]), else this equation has no solution in $\mathbb{F}_{2^m}$.

Let $T$ be a table containing elements of $\mathbb{F}_{2^m}$.

$$T[i] = \begin{cases} j & \text{if } j^2 + j = i \\ \emptyset & \text{if } \mathrm{Tr}(i) = 1 \end{cases}$$

In other words, $T[i]$ contains one of the two elements of the kernel of $f_i$, if $i$ is in the image of $x \mapsto x^2 + x$. Note that $j+1$ is the second element of this kernel.

Now, we read from the table, the element $T[b/a^2]$. We invert the change of variable by computing: $z_1 = ab$. Then, we compute the second solution: $z_2 = ab + a$. This process costs one multiplication and one addition in $\mathbb{F}_{2^m}$. Thus, we have solved our problem within four operations in $\mathbb{F}_{2^m}$ with a memory of $2^m$ field elements. It is useful for small $m$.

**Time-Memory Tradeoff for Degree 3.** In this case, the equation to solve is: $z^3 + az^2 + bz + c = 0$ where $a$, $b$, $c \in \mathbb{F}_{2^m}$. We obtain an affine multiple of degree 4 by multiplying the polynomial by $z + a$. It costs two multiplications, one squaring and two additions in $\mathbb{F}_{2^m}$. The substitution $z = \sqrt{(a^2 + b)}x$ ($m - 1$ repeated squarings[3]) and a normalization (two divisions) enable to obtain an equation of the form: $x^4 + x^2 + dx = e$, with $d$, $e \in \mathbb{F}_{2^m}$. Let us denote $f$ the mapping $x \mapsto x^4 + x^2 + dx$. By construction $a$ is a root of $f(x) = e$, we want to find the other three. The mapping $f$ has a kernel of dimension two (except if $d = 0$, in this case, the dimension is one). We have only to store two elements which form a basis of the kernel of $x \mapsto x^4 + x^2 + dx$ in a table for all $d \in \mathbb{F}_{2^m}$. This requires a storage memory of $2 \times 2^m$ field elements. Notice, this step does not depend on the coefficient $e$. Let us call the lookup table $T$ and the two elements stored in the table for a given $d$: $\lambda_1$ and $\lambda_2$. Then, we have $T[d] = (\lambda_1, \lambda_2)$. As $f$ is linear, the three other roots of $f$ are: $a+\lambda_1$, $a+\lambda_2$ and $a + \lambda_1 + \lambda_2$. We obtain them with three additions. Lastly, we invert the substitution (three multiplications) and thus the problem is solved. Here, we used the fact that we know that $a$ is a root of $f$ to find the other ones. We cannot use anymore this extra information for polynomials of degree 4 onwards.

[3] Let us mention that one addition and one multiplication with a constant, are enough using the method proposed in [Hub02]. We do not take it into account in Table 1.

### 4.3 How do we Compute Theoretical Complexity?

We will not give here the complexity recurrence formula, that we use to compute the number of operations required to process BTZ for the sake of clarity. Instead, we prefer to explain how we obtain it.

**About BTA.** The polynomials with which we deal in BTA are monic, without multiple root and can be factorized into linear factors over $\mathbb{F}_{2^m}$. These polynomials form a set $\mathcal{P}$. Such polynomials of degree $d$ are entirely determined by their $d$ roots. Hence, there are $\binom{2^m}{d}$ such polynomials.

Moreover, we know that for all $\beta \in B$, a $\mathbb{F}_2$-basis of $\mathbb{F}_{2^m}$, half of the elements of $\mathbb{F}_{2^m}$ have $\mathrm{Tr}(\beta \cdot z) = 0$ and that for the other half, $\mathrm{Tr}(\beta \cdot z) = 1$.

For each step of Algorithm 2 (see Appendix F), we compute the gcd of a polynomial with $\mathrm{Tr}(\beta \cdot z)$. The polynomial that we obtain contains the roots, such that $\mathrm{Tr}(\beta \cdot z) = 0$. We then make a Euclidean division that gives us another polynomial of degree $d - i$, which contains the roots, such that $\mathrm{Tr}(\beta \cdot z) = 1$. When we compute the theoretical complexity of BTZ, we compute the expected value of the number of operations in the worst case. Hence, we consider, for all $i \in \{0, \cdots, d\}$, the probability $P(d, m, i)$ that the polynomial of degree $d$ breaks down into a couple of polynomials of degree $i$ and $d-i$. That is, the gcd computation gives us a polynomial of degree $i$ whose roots are among the elements of $\mathbb{F}_{2^m}$, such that $\mathrm{Tr}(\beta \cdot z) = 0$. In the same manner, with the Euclidean division, we obtain a polynomial of degree $d-i$, while its roots are among the elements of $\mathbb{F}_{2^m}$, such that $\mathrm{Tr}(\beta \cdot z) = 1$. We assume the input polynomial is chosen at random from a uniform distribution over $\mathcal{P}$. Thus, we obtain:

$$P(d, m, i) = \frac{\binom{2^{m-1}}{i} \times \binom{2^{m-1}}{d-i}}{\binom{2^m}{d}}.$$

**About Zinoviev Procedures.** In Table 1, we assume that all the arithmetic operations on the field have unitary cost and that one binary matrix inversion of order $m$ costs $m^2$ additions in $\mathbb{F}_{2^m}$. Let $Z_d$ denote the Zinoviev procedure for degree $d$ where $d$ varies from 2 to 5. For $d = 2$ and $d = 3$, we consider two possibilities: with or without the time-memory tradeoff. When we use the tradeoff, the space complexity is exponential in m, that is, it is in the order of the size of the field *i.e.* $2^m$, up to a multiplicative constant factor. For greater degrees ($6 \leq d \leq 10$), the time complexity is $\mathcal{O}(m^2 + dm + d2^d)$. It is exponential in $d$ since in the worst case, the affine multiple has degree equal to $2^{d-1}$. Therefore, in the last step of Zinoviev

procedures, we would have to evaluate the polynomial in $2^{d-1}$ points for finding its roots. This is a reason for which we do not use Zinoviev methods for higher degrees than 10. One observes, in Table 1, that the most expensive part of the Zinoviev procedures is the binary matrix inversion which enables to find the roots of the affine polynomial (see Section 3.2 for more details).

| | Addition | Mult. | Division | Squaring | Matrix Inv. | Total Cost |
|---|---|---|---|---|---|---|
| $Z_2$ precomput. | $m^2$ | $m^2$ | 0 | $m(m-1)$ | 1 | $4m^2-m$ |
| $Z_2$ w/o tradeoff | $m$ | 1 | 1 | 1 | 0 | $m+3$ |
| $Z_2$ w/ tradeoff | 1 | 1 | 1 | 1 | 0 | 4 |
| $Z_3$ w/o tradeoff | $2(m+1)$ | $3m$ | 0 | $m$ | 1 | $m^2+6m+2$ |
| $Z_3$ w/ tradeoff | 5 | 5 | 2 | $m$ | 0 | $m+12$ |
| $Z_4$ | $2m+9$ | $3(m+1)$ | 8 | $m$ | 1 | $m^2+6m+20$ |
| $Z_5$ | $4m+101$ | $7m+104$ | 1 | 0 | 1 | $m^2+11m+206$ |

**Table 1.** Number of operations over $\mathbb{F}_{2^m}$ in Zinoviev procedures

## 5 Simulation Results

In Table 2, we provide experimental data for finding the roots of a polynomial of degree $t = 32$ over $\mathbb{F}_{2^m} = \mathbb{F}_{2048}$. $\text{BTZ}_d$ means that we use BTZ with $d_{max} = d$, for all suitable $d$.

| $n = 2048$, $t = 32$, $m = 11$ | | Chien | BTA | $\text{BTZ}_2$ | $\text{BTZ}_3$ | $\text{BTZ}_4$ |
|---|---|---|---|---|---|---|
| # CPU cycles per byte for | root finding | 3200 | 1300 | 900 | 800 | 800 |
| | decrypting | 3700 | 1800 | 1400 | 1300 | 1300 |
| percentage [4] of time spent for | syndrome computation | 5 | 10 | 13 | 14 | 14 |
| | solving key equation | 7 | 14 | 18 | 19 | 19 |
| | root finding | 87 | 72 | 65 | 61 | 60 |

**Table 2.** Experimental data for finding the roots of a polynomial of degree $t = 32$ over $\mathbb{F}_{2^m} = \mathbb{F}_{2048}$.

In Table 3, we present the theoretical number of field operations for correcting $t = 32$ errors of a $n = 2048$-length word over $\mathbb{F}_{2^{11}}$ using BTZ with $d_{max} = 5$ and time-memory tradeoff (see Section 4.2). For information, BTZ without the tradeoff gives very close *theoretical* results.

[4] Remaining percentages correspond to other minor tasks.

| $n = 2048$, $t = 32$, $m = 11$ | Chien | BTA | $\text{BTZ}_2$ | $\text{BTZ}_3$ | $\text{BTZ}_4$ | $\text{BTZ}_5$ | $\text{BTZ}_6$ |
|---|---|---|---|---|---|---|---|
| $K(+) = K(\times) = 1$ | 129k | 16k | 13k | 11k | 10k | 10k | 10k |
| $K(+) = 1$, $K(\times) = m$ | 764k | 91k | 65k | 54k | 50k | 47k | 48k |

**Table 3.** Theoretical number of field operations for correcting $t = 32$ errors of a $n = 2048$-length word over $\mathbb{F}_{2^{11}}$ using BTZ with $d_{max} = 5$ and time-memory tradeoff.

Let us depict the gain of BTZ over BTA and Chien procedure in terms of percentage of number of operations according to the parameter $d_{max}$ and the polynomial degree $t$. The results (given in Figure 1 and Figure 2) take into account the time-memory tradeoff for degree 2 and 3. For the sake of readibility and relevance in cryptographic applications, we restrict to the case $m = 11$, $K(\times) = m$, $t \leq 100$ and $d_{max} \leq 6$. Additionally, we present similar results for greater degrees in Appendix E. Nevertheless, we have also computed these results for $m = 8, 11, 12, 13, 14, 15, 16, 20, 30, 40$, $K(\times) = 1$, $t \leq 300$, $d_{max} \leq 10$ and for both *with* and *without* the tradeoff. These data give the information that the higher is $t$, the higher is the optimal $d_{max}$. One can deduce from the two following graphs that for $m = 11$ and $t = 32$, the recommended theoretical value for $d_{max}$ is 5. Indeed, we have a substantial gain of 46% over BTA and 93% over Chien method for this value of $d_{max}$.

**Acknowledgements:** We would like to express our gratitude to our PhD advisor, Nicolas Sendrier, for his valuable help during this work.

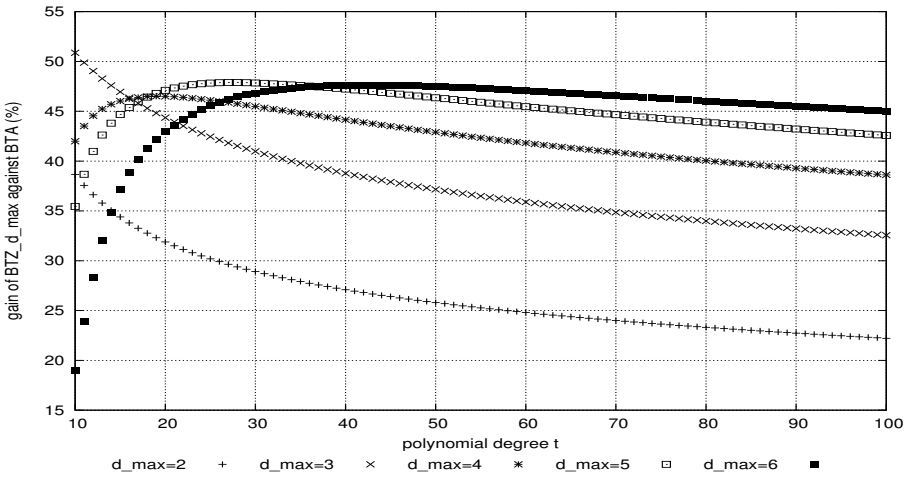


**Fig. 1.** $\mathrm{BTZ}_{d_{max}}$ vs. BTA; $m = 11$; $K(+) = 1$; $K(\times) = m$; with time-memory tradeoff

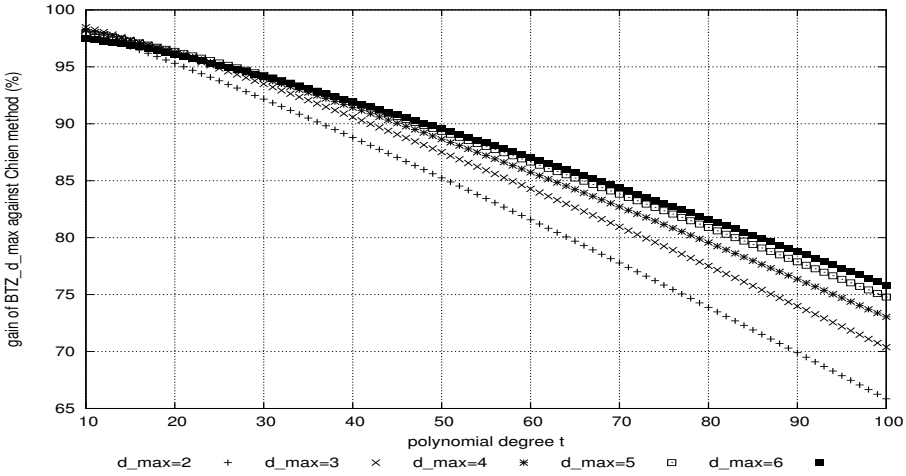


**Fig. 2.** $\mathrm{BTZ}_{d_{max}}$ vs. Chien; $m = 11$; $K(+) = 1$; $K(\times) = m$; with time-memory tradeoff

## References


[Ber71] E. R. Berlekamp. Factoring polynomials over large finite fields. In *SYMSAC '71 - Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, page 223, New York, USA, 1971. ACM.

[BRS67] E. R. Berlekamp, H. Rumsey, and G. Solomon. On the solution of algebraic equations over finite fields. In *Information and Control*, volume 10, pages 553–564, June 1967.

[BS08] Bhaskar Biswas and Nicolas Sendrier. McEliece cryptosystem implementation: Theory and Practice. In *PQCrypto*, pages 47–62, 2008.

[Chi64] R. T. Chien. Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes. In *IEEE Transactions on Information Theory*, volume 10, pages 357–363, 1964.

[FTC03] S. Fedorenko, P. Trifonov, and E. Costa. Improved hybrid algorithm for finding roots of error-locator polynomials. In *European Transactions on Telecommunications*, volume 14, pages 411–416, 2003.

[Gop70] V.D. Goppa. A new class of linear error-correcting codes. In *Probl. Inform. Transm.*, volume 6, pages 207–212, 1970.

[Hub02] K. Huber. Note on decoding binary Goppa codes. In *Electronics Letters*, volume 32, pages 102–103, August 2002.

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, October 1996.

[McE78] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. Technical Report DSN 42-44, JPL, Pasadena, 1978.

[MS83] F. J. Macwilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes (North-Holland Mathematical Library)*. North Holland, January 1983.

[MvOV88] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Some computational aspects of root finding in GF($q^m$). In *ISSAC*, pages 259–270, 1988.

[Pat75] N. Patterson. The algebraic decoding of Goppa codes. In *Information Theory, IEEE Transactions on*, volume 21, pages 203–207, 1975.

[vzGG03] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge Univ. Press, 2003.

[Zin96] V.A. Zinoviev. On the solution of equations of degree $\leq 10$ over finite fields GF($2^q$). In *Rapport de recherche INRIA $n^o$ 2829*, 1996.

z

## A Description of Classical McEliece Cryptosystem (1978)

**Public key:** A binary linear [n,k] code $C$, *i.e.* a $k$-dimensional linear $\mathbb{F}_2$-subspace of $\mathbb{F}_2^n$, described by a generator matrix $G$.

**Private key:** An efficient decoding algorithm for $C$ up to the error correcting capacity $t$.

**Encryption:** Map the $k$ bits plaintext $x$ to the codeword $x.G$, add $e$, an uniformly random error of length $n$ and weight $t$ to obtain the cyphertext $y$.

**Decryption:** Correct the $t$ errors, unmap to get the message. This process is also called decoding.

## B Description of Binary Goppa Codes (1970)

Let $m > 0$, $n \leq 2^m$ and $a = (a_1, ..., a_n) \in \mathbb{F}_2^n$.
The $n$-length binary Goppa code $\Gamma(L, g)$ is defined by:

- Support $L = (\alpha_1, ..., \alpha_n)$ $n$-tuple of distinct elements of $\mathbb{F}_{2^m}$;
- Goppa polynomial $g(z) \in \mathbb{F}_{2^m}[z]$, square-free, monic of degree $t > 0$ with no root in $L$.

In practice, $g$ is chosen irreducible over $\mathbb{F}_{2^m}$ and $n = 2^m$.
$\Gamma(L, g)$ is a subfield subcode over $\mathbb{F}_2$ (*i.e.* the subcode that contains all the codewords whose coordinates are in $\mathbb{F}_2$) of a particular Goppa geometric code over the binary field $\mathbb{F}_{2^m}$.
We have $a \in \Gamma(L, g)$ if and only if:

$$R_a(z) := \sum_{i=1}^{n} \frac{a_i}{z - \alpha_i} = 0 \text{ over } \mathbb{F}_{2^m}[z]/(g(z)).$$

$R_a$ is called the syndrome of the word $a$. Note that $R_a$ is $\mathbb{F}_2$-linear in $a$.
*Binary* Goppa codes have an error correction capacity of at least $t$ errors. We present a polynomial-time algorithm for decoding these codes that corrects up to $t$ errors.

**Algebraic Decoding Algorithm**

Let $e$, $x$, $y$ be $n$-length binary vectors. We have to find $x$, the sent codeword from $y = x + e$ where $y$ is the received word and $e$, the error word.

Algebraic decoding is carried out in three steps:

1. Syndrome computation of the received word
$$R_y(z) = R_e(z) = \sum_{i=1}^{n} \frac{e_i}{z - \alpha_i} \text{ over } \mathbb{F}_{2^m}[z]/(g(z)).$$
2. Solving the key equation to obtain the *error locator polynomial* $\sigma_e$ with the Patterson algorithm [Pat75].
$$R_e(z) \cdot \sigma_e(z) = \sigma_e'(z) \text{ over } \mathbb{F}_{2^m}[z]/(g(z)).$$
Notation: $\sigma_e'$ denotes the formal derivative of $\sigma_e$.
3. Error locator polynomial root finding
$$\sigma_e(z) := \prod_{i=1}^{n} (z - \alpha_i)^{e_i};\ e_i \neq 0 \Leftrightarrow \sigma_e(\alpha_i) = 0.$$
We stress on the fact that the degree of $\sigma_e$ is also the Hamming weight of $e$, that is to say, the number of errors to correct. The algorithm can correct up to $t$ errors so the degree of $\sigma_e$ is less than or equal to $t$.

## C Description of Chien Procedure (1964)

Chien search is a recursive algorithm. It is a clever exhaustive search. Let $f(x) = a_0 + a_1 \cdot x + \cdots + a_t \cdot x^t$ be a polynomial over $\mathbb{F}_{2^m}$ and let $\alpha$ be a generator of the multiplicative group $\mathbb{F}_{2^m}^*$.

$$\begin{aligned} f(\alpha^i) &= a_0 + a_1 \cdot \alpha^i + \cdots + a_t \cdot (\alpha^i)^t \\ f(\alpha^{i+1}) &= a_0 + a_1 \cdot \alpha^{i+1} + \cdots + a_t \cdot (\alpha^{i+1})^t \\ &= a_0 + a_1 \cdot \alpha^i \cdot \alpha + \cdots + a_t \cdot (\alpha^i)^t \cdot \alpha^t \end{aligned}$$

Set $a_{i,j} = a_j(\alpha^i)^j$. It is easy to obtain $f(\alpha^{i+1})$ from $f(\alpha^i)$ since we have that $a_{i+1,j} = a_{i,j} \cdot \alpha^j$. Moreover, if $\sum_{j=0}^{t} a_{i,j} = 0$, then $\alpha^i$ is a root of $f$.

## D Finding Roots of an Affine Polynomial

Let us have an affine polynomial $A(z) = L(z) + c = \sum_{i=0}^{m-1} l_i \cdot z^{2^i} + c$. Consider $(\alpha_1, \cdots, \alpha_m)$ is a $\mathbb{F}_2$-basis of $\mathbb{F}_{2^m}$, $(l_i)_{1 \le i \le m}$, $c$ and $x$ are elements of $\mathbb{F}_{2^m}$. Guess $x = (x_1, \cdots, x_m)$ is a root of $A$. Finding zeroes of an affine polynomial is equivalent to solving a linear system. Indeed, we have:

$$
\begin{aligned}
A(x) = 0 \Leftrightarrow L(x) &= c \\
&\Leftrightarrow \sum_{i=1}^{m} x_i \cdot L(\alpha_i) = \sum_{i=1}^{m} c_i \cdot \alpha_i \quad \text{(using linearity of } L\text{)} \\
&\Leftrightarrow \sum_{i=1}^{m} \sum_{j=1}^{m} x_i \; l_{i,j} \cdot \alpha_i = \sum_{i=1}^{m} c_i \cdot \alpha_i \quad \text{(linear system in } x_i\text{)}.
\end{aligned}
$$

## E $\mathrm{BTZ}_{d_{max}}$ vs. BTA for $m = 11$, $30 \le t \le 300$

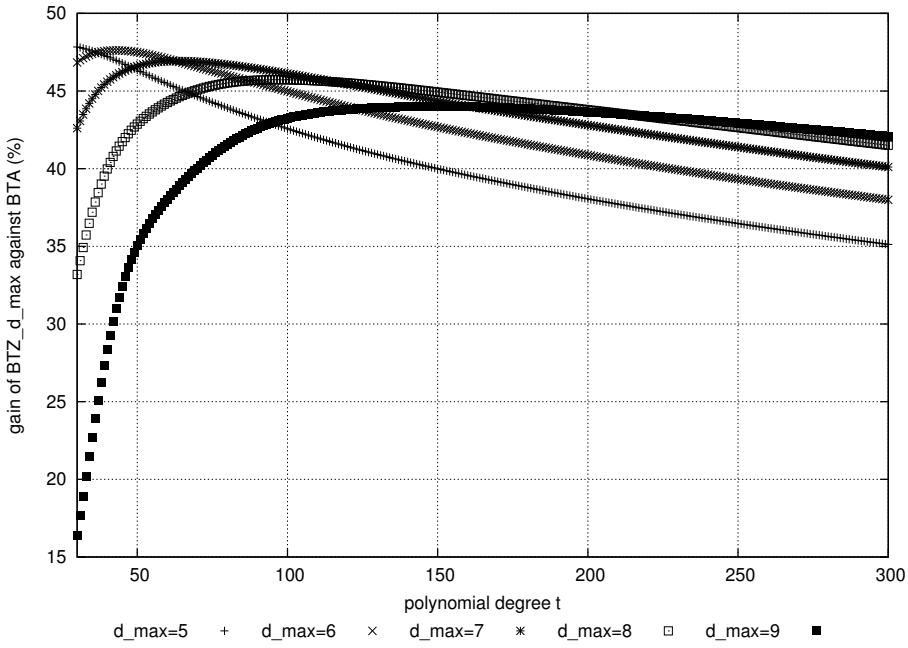


**Fig. 3.** $\mathrm{BTZ}_{d_{max}}$ vs. BTA; $m = 11$; $K(+) = 1$; $K(\times) = m$; with time-memory tradeoff

## F Pseudocodes of BTZ

Some notations:

- $\sigma_e$ is the error locator polynomial of the error word $e$.
- $d_{max}$ is the maximum degree up to which we use Zinoviev procedures.
- $D_{Zin}$ is the set of degrees for which we apply Zinoviev procedures.
- $(\beta_1, \cdots, \beta_m)=(\alpha, \alpha^2, \cdots, \alpha^m)$ is a fixed polynomial basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ where $\alpha$ is a primitive element of $\mathbb{F}_{2^m}$.

**Algorithm 1** simplified BTZ without precomputation - BTZ$(f, d, i)$

First call: $f \leftarrow \sigma_e$; $d \leftarrow d_{max} \in \{2, \ldots, 10\}$; $i \leftarrow 1$.
**if** degree$(f) \leq d$ **then**
  **return** ZINOVIEV$(f, d)$;
**else**
  $g \leftarrow \gcd(f, \mathrm{Tr}(\beta_i \cdot z))$;
  $h \leftarrow f/g$;
  **return** BTZ$(g, d, i+1) \cup$ BTZ$(h, d, i+1)$;
**end if**

**Algorithm 2** BTZ with precomputation - BTZ$(f, D, i)$

First call: $f \leftarrow \sigma_e$; $D \leftarrow D_{Zin} \subset \{2, \ldots, 10\}$; $i \leftarrow 1$.
{precomputation phase}
**for** $1 \leq i \leq m$ **do**
  $T_i \leftarrow \mathrm{Tr}(\beta_i \cdot z)) \mod f$;
**end for**
$i \leftarrow 1$;
{computation phase}
**if** degree$(f) \in D$ **then**
  **return** ZINOVIEV$(f, d)$;
**else**
  $T \leftarrow T_i \mod f$;
  $g \leftarrow \gcd(f, T)$;
  $h \leftarrow f/g$;
  $i \leftarrow i+1$;
  **return** BTZ$(g, d, T) \cup$ BTZ$(h, d, T)$;
**end if**