Computing p*th* roots in extended finite fields of prime characteristic $p \geq 2$

M. Repka

Direct computation of p*th* roots in extended finite fields of characteristic $p \geq 2$ is introduced, wherein the reduction polynomial is irreducible and can be even random. Proposed method works in any case of $p \geq 2$ and finite field extension. This method is the most efficient, it is even more efficient than the method, which is widely used, based on inversion of squaring matrix utilised in the case of $p = 2$. This method is more efficient regarding the computation and storing of the matrix as well as the computation of the roots.

Introduction: Let $\mathbb{GF}(p^m) := \mathbb{GF}(p)[X]/m(X)$ be the finite subfield, where $m(X)$ is an irreducible polynomial of $\deg m(X) = m > 0$, and p is a prime ≥ 2.

Let $g(Z)$, $\deg g(Z) = t > 0$, be an irreducible polynomial in $\mathbb{GF}(p^m)[Z]$, then the extended finite field is defined as $\mathbb{GF}(p^{mt}) := \mathbb{GF}(p^m)[Z]/g(Z)$.

This work deals with computation of p*th* root $r(Z)$ of a polynomial $n(Z) \in \mathbb{GF}(p^m)[Z]$, $0 \leq \deg n(Z) \leq t-1$, modulo the $g(Z)$:

$$r(Z) \equiv \sqrt[p]{n(Z)} \bmod g(Z), \tag{1}$$

where $r(Z)$ and $n(Z)$ are polynomial representations of elements in the extended finite field $\mathbb{GF}(p^{mt})$.

This kind of p*th* root computation is very important in many applications, particularly in coding and cryptography. One interesting example is the original [1] McEliece PKC which is proposed to use binary irreducible goppa codes, or the Niederreiter PKC based on the same codes, which belongs to the post-quantum cryptography, cryptography which is not threaten even employing quantum cryptanalysis.

Related Work & Our Contribution: Therefore, there exist alternatives how to compute the p*th* root. The most efficient and widely used method in case of $p = 2$ is to use so called square root matrix created taking inversion [2] of squaring matrix $\mathbf{S}_{t\times t}$ defined in [3]. Comparisons of existing methods for case $p = 2$ can be found in [4].

In a case of $p > 2$, for instance modular inversion of powering matrix, discrete logarithm of a primitive element can be exploited. For $p \geq 5$, there is an interesting method [5] working only in certain cases of the reduction polynomial. Further methods for certain cases are in [6].

This method is a trade-off between computations and memory. Computing roots in $\mathbb{GF}(p^m)[Z]/g(Z)$ is split into a matrix multiplication and computing roots in $\mathbb{GF}(p^m)$, such as in the case of the square root matrix which works for $p = 2$ only. However, even comparing to the widely used method using $\mathbf{S}^{-1}_{t\times t}$, this method is more efficient regarding both the matrix preparation and storing, and also regarding the computation of roots. The preparation of the matrix (17) is more efficient because it has less columns, because of the (11) that computes next carry polynomials based on initial carry polynomials in (9) and then based on previous ones, and particularly because there is not computed the inversion. The final roots computation is more efficient because the matrix (17) has less columns. The matrix in (17) is $t \times t - \lceil t/p \rceil$, see (3), (6), and (17) respectively. Hence, less computations and memory is needed to create and store the matrix, and fewer multiplications and additions are needed to compute the roots.

*The p*th *root computation:* The p*th* root (1) is split to the **L**eft and **R**ight parts, in the way **L** contains exponents of terms which do not have to be reduced, and **R** contains exponents of carry terms which must be reduced, as follows

$$\mathbf{L} := (L_j)_{0\leq j<t_1}, \tag{2}$$

$$t_1 := \lceil t/p \rceil, \tag{3}$$

$$L_j := jp, \tag{4}$$

$$\mathbf{R} := (R_k)_{0\leq k<t_2}, \tag{5}$$

$$t_2 := t - t_1. \tag{6}$$

For $k : 0 \leq k \leq p-2$, $k < t_2$:

$$R_k := (k+1)\pi, \tag{7}$$

$$\pi := p^{mt-1} \equiv p^{-1} \pmod{p^{mt}-1}, \tag{8}$$

$$c_{R_k}(Z) \equiv Z^{R_k} \bmod g(Z), \tag{9}$$

and for $k : (p-1) \leq k < t_2$:

$$R_k := R_{k-(p-1)} + 1. \tag{10}$$

$$c_{R_k}(Z) \equiv c_{R_{k-(p-1)}}(Z) Z \bmod g(Z). \tag{11}$$

The p*th* root can then be written as:

$$\sqrt[p]{n(Z)} \equiv \sum_{j=0}^{t_1-1} Z^j \sqrt[p]{n_{L_j}} + \sum_{k=0}^{t_2-1} Z^{R_k} \sqrt[p]{n_{\lambda(k)}} \bmod g(Z), \tag{12}$$

$$\lambda(k) := k + 1 + \lfloor k/(p-1) \rfloor. \tag{13}$$

The carry terms, which must be reduced, are substituted for the corresponding carry polynomials in (9) and (11) respectively. Therefore, (1) can be written as

$$r(Z) = \sum_{j=0}^{t_1-1} Z^j n_{L_j}^{p^{-1}} + \sum_{k=0}^{t_2-1} c_{R_k}(Z) n_{\lambda(k)}^{p^{-1}} \tag{14}$$

if represented as polynomials, or as (15) if represented as vectors

$$\mathbf{r} = \mathbf{r_L} + \mathbf{r_R}, \tag{15}$$

$$\mathbf{r_L} := (r_i)_{0\leq i<t}, \quad r_i = \begin{cases} n_{L_i}^{p^{-1}} & \text{if } i < t_1 \\ 0 & \text{if } i \geq t_1 \end{cases}, \tag{16}$$

$$\mathbf{r_R} := \begin{array}{c} \\ Z^0 \\ \vdots \\ Z^{t_1-1} \\ \vdots \\ Z^{t-1} \end{array} \begin{array}{c} \begin{array}{cccc} Z^{R_0} & Z^{R_1} & \dots & Z^{R_{t_2-1}} \end{array} \\ \begin{pmatrix} \mathbf{c_{R_0}} & \mathbf{c_{R_1}} & \cdots & \mathbf{c_{R_{t_2-1}}} \end{pmatrix} \end{array} \begin{pmatrix} n_{\lambda(0)}^{p^{-1}} \\ n_{\lambda(1)}^{p^{-1}} \\ \vdots \\ n_{\lambda(t_2-1)}^{p^{-1}} \end{pmatrix}, \tag{17}$$

where $\mathbf{c_{R_k}}$ are column vectors of coefficients of polynomials $c_{R_k}(Z)$. Multiplications, additions, and pth roots are operations in the subfield.

Conclusion: Very efficient, direct p*th* root computation in extended finite fields of characteristic $p \geq 2$ working even for random irreducible reduction polynomial and for any finite field extension was proposed. The matrix in (17) is $t \times t_2$, see (3) and (6), which is less than in $\mathbf{S}^{-1}_{t\times t}$ case. Further, the inversion is not needed here, this is indeed direct computation. Carry polynomials in (11) are computed efficiently using initial (9) and then previous carry polynomials, thus big exponents in (10) are not used, it is only important to consider k instead of R_k. Fewer multiplications and additions are needed to compute the roots, less computations and memory is needed to create and store the matrix, and if parameters p, $m(X)$, and t are fixed, this solution can be hardwired, only the matrix elements must be recomputed if $g(Z)$ change.

Acknowledgment: Supported in part by NATO's Public Diplomacy Division in the framework of "Science for Peace", SPS Project 98452.

M. Repka (*Dept. of Applied Informatics and Information Tech., Inst. of Computer Science and Math., FEI STU, Ilkovičova 3, Bratislava, SK-812 19, SVK.*)
E-mail: marek.repka@stuba.sk

References

1 Repka, M.: 'McEliece PKC Calculator'. *Journal of Electrical Engineering*, 2014, vol. **65**(6), pp 342-348.
2 Strenzke, F.: 'A Smart Card Implementation of the McEliece PKC'. *WISTP 2010, Passau, Germany, Proceedings*, 2010, pp 47-59.
3 IEEE Std 1363-2000 Working Group. 'IEEE 1363: Standard Specifications for Public-Key Cryptography'. *IEEE, Inc., USA*, 2000.
4 Doliskani, J., Schost, E.: 'Computing in degree 2^k-extensions of finite fields of odd characteristic'. *Des. Code Cryptogr*, 2015, vol. **74**(3), pp 559-569.
5 Panario, D., Thomson, D.: 'Efficient pth root computations in finite fields of characteristic p'. *Des. Code Cryptogr*, 2009, vol. **50**(3), pp 351-358.
6 Barreto, P. S. L. M and Voloch, J. F.: 'Efficient Computation of Roots in Finite Fields'. *Design Code Cryptogr*, 2006, vol. **39**(2), pp 275-280.