DE GRUYTER OPEN

Journal of ELECTRICAL ENGINEERING, VOL 67 (2016), NO1, 56–60

NOTE ON MODULAR REDUCTION IN EXTENDED FINITE FIELDS AND POLYNOMIAL RINGS FOR SIMPLE HARDWARE

Marek Repka *

Modular reduction in extended finite fields and polynomial rings is presented, which once implemented works for any random reduction polynomial without changes of the hardware. It is possible to reduce polynomials of whatever degree. Based on the principal defined, two example RTL architectures are designed, and some useful features are noted furthermore. The first architecture is sequential and reduce whatever degree polynomials, taking 2 cycles per term. The second one is Parallel and designed for reduction of polynomials of $2(t\text{-}1)$ degree at most, taking 1 cycle for the whole reduction.

K e y w o r d s: modular reduction, extended finite fields, extended polynomial rings, algebraic codes, McEliece PKC, Niederreiter PKC

1 INTRODUCTION

Many methods are optimized, designed, and implemented for special cases of finite fields, or special cases of reduction polynomial that must be of a certain constant value, and, in better cases, of a certain form. This general reduction works for any reduction polynomial, for instance [1–6], and thus we can say that it works for random reduction polynomials which can be even secret [7–11]. No special methods are used, and thus no processor is required, just a simple hardware can be used. The main contributions of this paper are the generalized mathematical description, and the RTL schemes.

2 PROBLEM DEFINITION & NOTATION

Let $\mathbb{GF}(p^m) = \mathbb{GF}(p)[X]/m(X)$ be the finite field, where $m(X)$ is irreducible of $\deg m(X) = m > 0$, and p is a prime ≥ 2. The polynomial $m(X)$ is chosen by a designer, and it is a constant parameter. Thus also the operations $\otimes$ and $\oplus$ can be optimized and hardwired (binary case example [12]), as we consider it is done hereafter. Let an element in $\mathbb{GF}(p^m)$ is represented using h wires or bits.

$$h = \lfloor \log_2 (p^m - 1) \rfloor + 1.$$

Our aim is to compute remainder $r(Z)$ in the $\mathbb{GF}(p^m)[Z]$ modulo a reduction polynomial $g(Z)$ that is not hardwired and can be changed without any changes of the hardware.

$$r(Z) \equiv n(Z) \bmod g(Z)\,, \tag{1}$$

where $n(Z)$ is a polynomial which remainder of is going to be determined, $\deg n(Z) > t - 1$.

For the reason $g(Z)$ is a variable parameter, operations $\bmod g(Z)$ cannot be hardwired as it is in case of $\otimes$ and $\oplus$. For the computation of inversions $(-g_t)^{-1}$ consult [13]. If the reduction polynomial $g(Z)$ is monic, the inversion is not important to compute as the inversion of 1 is still 1, and if the application uses only monic polynomials, such as McEliece PKC, the inversion do not have to be implemented.

3 THE PRINCIPAL

The general modular reduction is based on the following fact

$$g(Z) = g_t Z^t \oplus g_{t-1} Z^{t-1} \oplus \ldots \oplus g_0 = 0\,, \tag{2}$$

$$Z^t = (-g_t)^{-1} \left(g_{t-1} Z^{t-1} \oplus \ldots \oplus g_0 \right)\,. \tag{3}$$

The inversion $(-g_t)^{-1}$ and the multiplications by the inversion are in the $\mathbb{GF}(p^m)$, thus after adjustments, polynomial $c_t(Z)$ is obtained:

$$c_t(Z) = g_{t-1}^{(0)} Z^{t-1} \oplus \ldots \oplus g_0^{(0)}, \tag{4}$$

$$Z^t \equiv c_t(Z) \bmod g(Z)\,. \tag{5}$$

For $i \geq 1$, we can shift to left:

$$Z^{t+i} = c_{t+i-1}(Z) Z\,, \tag{6}$$

$$Z^{t+i} = g_{t-1}^{(i-1)} Z^t \oplus \ldots \oplus g_0^{(i-1)} Z\,, \tag{7}$$

and by substituting Z^t for $c_t(Z)$, we get

$$c_{t+i}(Z) = g_{t-1}^{(i)} Z^{t-1} \oplus \ldots \oplus g_0^{(i)}, \tag{8}$$

$$Z^{t+i} \equiv c_{t+i}(Z) \bmod g(Z)\,. \tag{9}$$

* Institute of Computer Science and Mathematics, Faculty of Electrical Engineering and Information Technology, Ilkovičova 3, Bratislava, SK-812 19, Slovakia, marek.repka@stuba.sk

DOI: 10.1515/jee-2016-0008, Print (till 2015) ISSN 1335-3632, On-line ISSN 1339-309X

We call carry terms the Z^{t+i} terms, and we call the $c_{t+i}(Z)$ as their carry polynomials.

4 THE PRINCIPAL IN GENERAL MODULAR REDUCTION

A polynomial $n(Z)$ the remainder of which is going to be determined can be written as

$$n(Z) = n_L(Z) \oplus n_R(Z) = \bigoplus_{i=0}^{n-t} n_{t+i} Z^{t+i} \oplus \bigoplus_{i=0}^{t-1} n_i Z^i. \quad (10)$$

The terms the polynomial $n_L(Z)$ consists of must be reduced and added to the $n_R(Z)$. First, coefficient n_t is taken and multiplied by the carry polynomial $c_t(Z)$ of its carry term Z^t, see (5), which results to an intermediate polynomial $r_I^{(t)}(Z)$ of deg at most $t-1$. These intermediate polynomials are essentially the reduced terms into polynomials which are added to the $n_R(Z)$. Polynomial $r_I^{(t)}(Z)$ is then added to the $r^{(0)}(Z)$ resulting to $r^{(1)}(Z)$. Accordingly, next coefficient n_{t+1} is multiplied by carry polynomial $c_{t+1}(Z)$ of its carry term Z^{t+1} resulting to $r_I^{(t+1)}(Z)$. The polynomial $r_I^{(t+1)}(Z)$ is added to $r^{(1)}(Z)$ resulting to $r^{(2)}(Z)$. In general, for $i \geq 0$ holds

$$r_I^{(t+i)}(Z) = n_{t+i} c_{t+i}(Z) \equiv n_{t+i} Z^{t+i} \bmod g(Z), \quad (11)$$

$$r^{(0)}(Z) = n_R(Z), \quad (12)$$

$$r^{(i+1)}(Z) = r_I^{(t+i)}(Z) \oplus r^{(i)}(Z). \quad (13)$$

The result $r(Z)$ of the modular reduction can be obtained as:

$$r(Z) = \bigoplus_{i=0}^{n-t} r_I^{(t+i)}(Z) \oplus n_R(Z) \quad (14)$$

or also

$$r(Z) = r^{(n-t+1)}(Z). \quad (15)$$

If we can upper bound the maximal degree of polynomials $n(Z)$, for an instance as $\max \deg n(Z) = n \leq (p^{mt}-1)(t-1)$, then the principal can be written as the following matrix vector product:

$$\begin{pmatrix} n_0 & g_0^{(0)} & \cdots & g_0^{(n-t)} \\ n_1 & g_1^{(0)} & \cdots & g_1^{(n-t)} \\ \vdots & \vdots & \ddots & \vdots \\ n_{t-1} & g_{t-1}^{(0)} & \cdots & g_{t-1}^{(n-t)} \end{pmatrix} \begin{pmatrix} 1 \\ n_t \\ n_{t+1} \\ \vdots \\ n_n \end{pmatrix} = \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_{t-1} \end{pmatrix}. \quad (16)$$

The matrix has t rows and $n-t+1$ columns. The resultant vector has t elements which in polynomial representation stands for the remainder.

Regarding the (5)–(15), serial reduction architecture (SRA) is designed in the Section SRA RTL Architecture. According to the SRA and considering (16), also Parallel Reduction Architecture (PRA) can be designed, what means that the carry polynomials are precomputed and memorized, such as example in the Section PRA RTL Architecture.

5 SOME OTHER FEATURES

While we are going to left (it is a special -but simple- shift-and-substitute operation) in revealing $c_{t+i}(Z)$, (6)–(9); we must go to right when revealing carry polynomials of 1 and Z^{-i}.

Shift coefficients of a $g(Z)$ right until the first nonzero coefficient is shifted out. Let this coefficient has index j, *ie* $g_j Z^j$. Let the following nonzero coefficient has index l, *ie* $g_l Z^l$. Note, $l > j \geq 0$.

$$Z^{-1} \equiv c_{-1}(Z) \bmod g(Z), \quad (17)$$

$$c_{-1}(Z) = -\bigoplus_{k=l}^{t} \frac{g_k}{g_j} Z^{k-j-1}. \quad (18)$$

$$Z^{-i} \equiv c_{-i}(Z) \bmod g(Z), \quad (19)$$

$$c_{-i}(Z) = c_{-i+1}(Z) Z^{-1}. \quad (20)$$

In the (17), only positive exponents are present. In the (20), the multiplication is the right shift, and if a negative exponent appears after the shift, it is Z^{-1}, and this term is substituted for $c_{-1}(Z)$, similarly as in (6)–(9). Further interesting features:

$$1 = Z^0 \equiv c_0(Z) \bmod g(Z), \quad (21)$$

$$c_0(Z) = -\bigoplus_{k=l}^{t} \frac{g_k}{g_j} Z^{k-j}. \quad (22)$$

$$n(Z) \mid c_0(Z) \iff \left(n^{-1}(Z) \bmod g(Z)\right) = \frac{c_0(Z)}{n(Z)}. \quad (23)$$

6 SRA RTL ARCHITECTURE

The architecture for SRA is depicted in Fig. 1. Since it is iterative architecture, polynomials of whatever degree can be reduced. This architecture can be enhanced to implement the special shift-and-substitute operation to right in order to perform some of the other features.

6.1 Initialization

In the 1st cycle, the $r^{(i+1)}(Z)$ memory block registers value of $n_R(Z)$ polynomial coefficients, and the Z^t memory block registers the first t coefficients of $g(Z)$. The carry polynomial of Z^t term is computed in the way that the actual value of the Z^t memory block is multiplied by $(-g_t)^{-1}$. This results to the carry polynomial of the Z^t carry term.

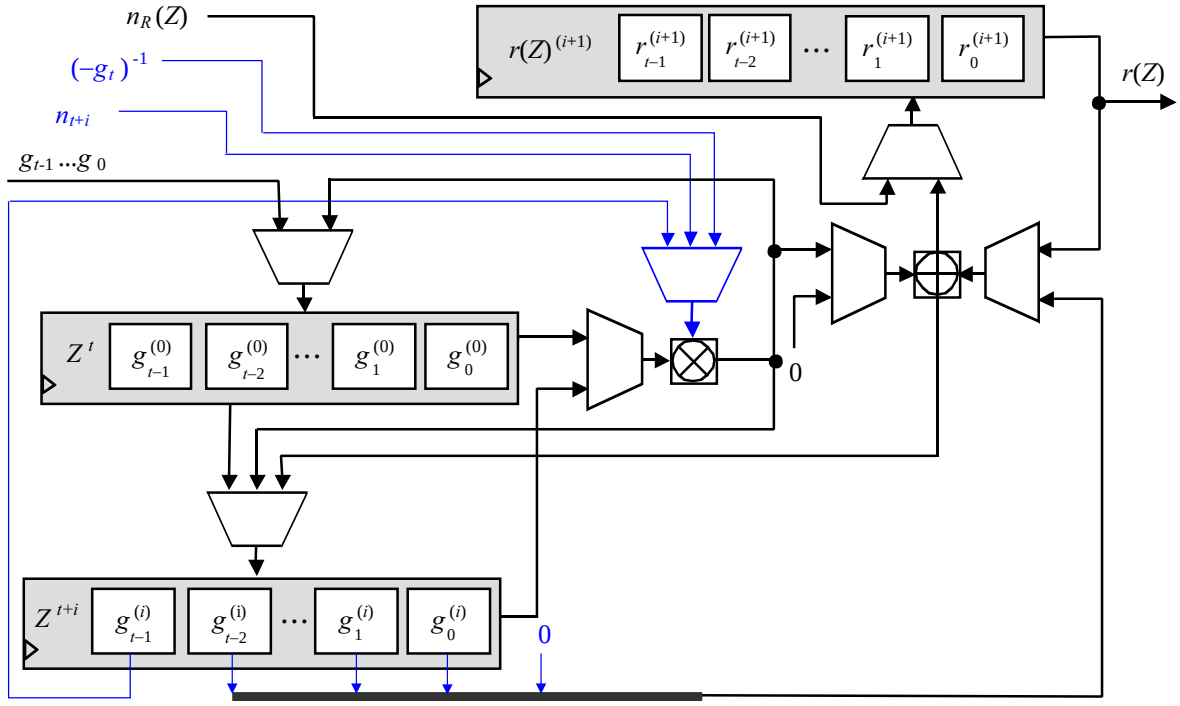

Fig. 1. RTL Architecture for SRA

6.2 Reduction

The carry polynomial is registered by the Z^t and the Z^{t+i} memory blocks in the 1st cycle. We have $i = 0$ in this cycle. Thus, in this cycle, n_{t+0} is multiplied by the value of the Z^{t+i} memory block. This results to the intermediate (reduced) polynomial $r_I^{(0)}(Z)$ that is added to the $r^{(0)}(Z)$ resulting into the $r^{(1)}(Z)$ that is registered by the $r^{(i+1)}(Z)$ memory block in the 2nd cycle.

In the 2nd cycle, $g_{t-1}^{(0)}$ is taken of the Z^{t+i} memory block, and it is multiplied by the value of the Z^t memory block. This product is added to the $[g_{t-2}^{(0)}, \ldots, g_0^{(0)}, e]$ vector that is also taken of the Z^{t+i} memory block with e added to the right (m LSBs). This results into the next carry polynomial that is registered by the Z^{t+i} in the 3rd cycle.

In the 3rd cycle, $i = 1$. Therefore, in this cycle, n_{t+1} is multiplied by the value of the Z^{t+i} memory block resulting to the $r_I^{(1)}(Z)$ that is added to the $r^{(1)}(Z)$ resulting into the $r^{(2)}(Z)$ that is registered by the $r^{(i+1)}(Z)$ memory block in the 4th cycle.

Consequently, in the 4th cycle, $g_{t-1}^{(1)}$ is taken of the Z^{t+i} memory block, and it is multiplied by the value of the Z^t memory block. This product is added to the $[g_{t-2}^{(1)}, \ldots, g_0^{(1)}, e]$ value. The result is the next carry polynomial that is registered by the Z^{t+i} in the 5^{th} cycle, which in turn makes SRA ready for reduction of the next term.

Since the carry polynomial of the Z^t carry term is stored in the Z^t memory block, when the reduction of next polynomial takes place, it is only important to register this carry polynomial by the Z^{t+1} memory block such as it was made in the 1st cycle, and continue from that point to finish the next reduction. Of course, if the next reduction is modulo another polynomial, the SRA must be reinitialized by the new values.

7 PRA RTL ARCHITECTURE

The top level architecture for PRA is depicted in Fig. 2, and its functional description follows. This architecture is rolled out for reduction of multiplication results, *eg* maximal degree $2(t-1)$, but it can be enhanced for both reduction of higher terms in parallel or sequential reduction of blocks of terms (a trade of between memory and number of cycles).

7.1 Initialization

In the 1st cycle, the Z^t memory block registers the first t coefficients of $g(Z)$. The carry polynomial of Z^t term is computed as the actual value of the Z^t memory block multiplied by $(-g_t)^{-1}$. This results to the carry polynomial of the Z^t carry term.

In the 2nd cycle, this value is registered by both the Z^t and the $Z^{2(t-1)}$ memory blocks. The next carry polynomial of the next carry term is computed by taking $g_{t-1}^{(0)}$ of the $Z^{2(t-1)}$ memory block, and multiplying it

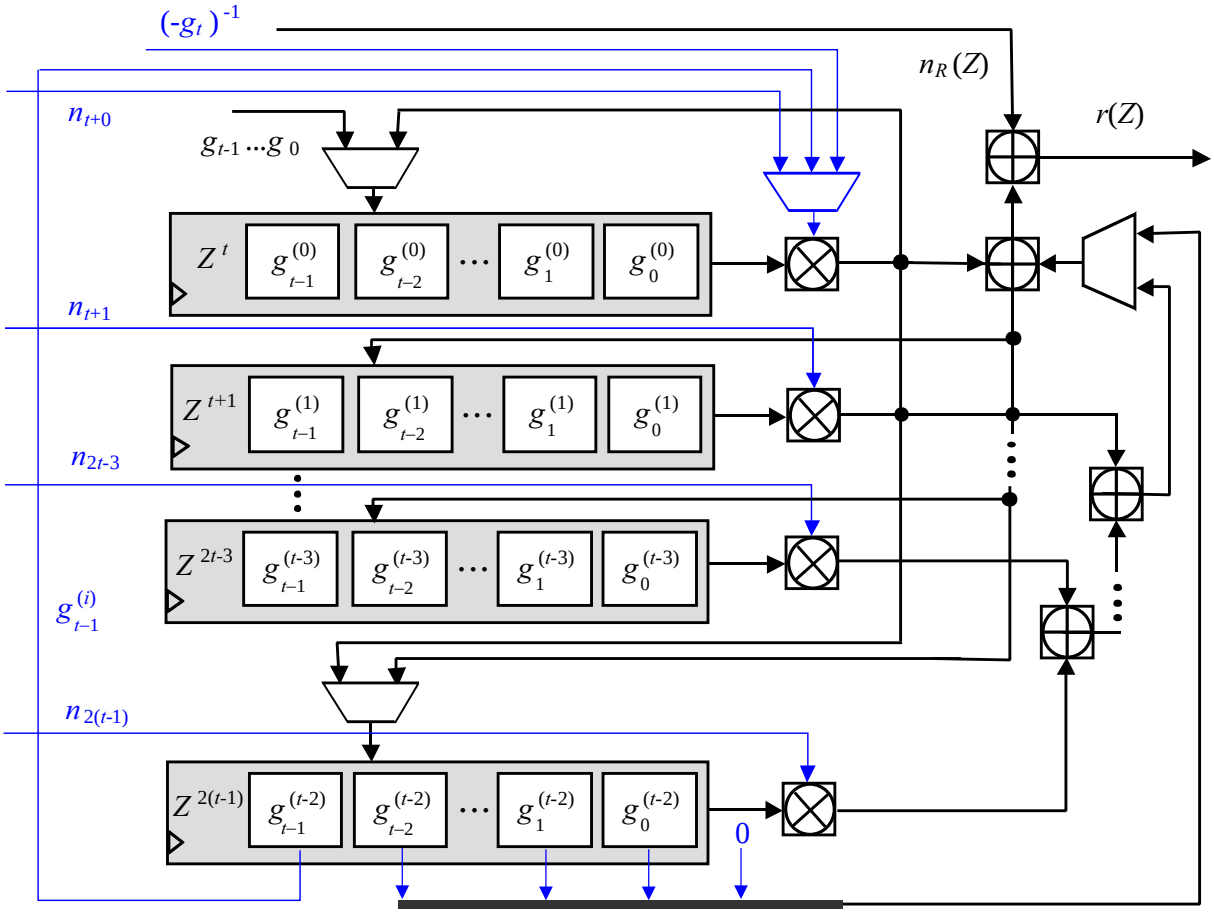

Fig. 2. RTL Architecture for PRA

by the value of the Z^t memory block. Result is added to $[g^{(0)}_{t-2} \ldots g^{(0)}_0 \; e]$, which is taken of $Z^{2(t-1)}$ memory block, resulting to the next carry polynomial.

The carry polynomial is registered by both the Z^{t+1} and the $Z^{2(t-1)}$ memory blocks in the 3rd cycle. Consequently, $g^{(1)}_{t-1}$ is taken of the $Z^{2(t-1)}$ memory block, and it is multiplied by the value of the Z^t memory block. This product is added to the $[g^{(1)}_{t-2} \ldots g^{(1)}_0 \; e]$ value, which results to the next carry polynomial that can be registered by the corresponding memory block, and $Z^{2(t-1)}$ memory block, in the following cycle.

In this manner, all the carry polynomials are computed and registered finishing by the value of the $Z^{2(t-1)}$ memory block. At the cycle these values have been registered, the reduction can begin. It is t^{th} cycle.

7.2 Reduction

This REM operation of Polynomials takes one cycle. All the n_{t+i} are taken and multiplied by their corresponding carry polynomials resulting to the reduced intermediate polynomials which are finally added together with the $n_R(Z)$ resulting to the final result $r(Z)$.

In the following cycle, the next reduction can be performed. However, if the reduction polynomial for the next reduction is different, PRA must be reinitialized.

8 BASIC EVALUATION OF COSTS AND PERFORMANCE

This really depends on the technology and platform used, and the representation of the $\oplus$ and $\otimes$ operations. For instance, the memory blocks can be implemented as registers, or as a part of embedded or external (SD Card) memory. Another example are the operations $\oplus$ and $\otimes$. These can be hardwired, or implemented based on log-and-anti-log and Zech's log techniques.

PRA can be designed to reduce parallel blocks of carry terms regarding the memory possibilities. The PRA RTL architecture depicted in Fig. 2 is an example where the degree of the polynomial to reduce is bounded by $2(t-1)$ which is the maximal degree of a product.

SRA needs $3ht$ bits in memory, 1 cycle to initialize, and 2 cycles per term reduction; while PRA needs 1 cycle per the whole polynomial reduction but $(t-1)th$ bits of memory. Further, an efficient trade-off can be considered,

and some improvements in performance can be achieved, for example utilizing [14].

Table 1. Basic evaluation of costs

RTL Architecture	Memory [bits]	MUX 3 : 1 [qty]	MUX 2 : 1 [qty]	$\otimes$	$\oplus$
SRA	$3th$	2	5	t	t
PRA - bounded by $\max\deg\ 2(t-1)$	$(t-1)th$	1	3	$(t-1)t$	$(t-1)t$

Table 2. The basic evaluation of performance

RTL Architecture	Initialization [cycles]	reduction [cycles]
SRA	1	2 per term
PRA - bounded by $\max\deg\ 2(t-1)$	t	1

9 CONCLUSION

We have described general reduction in random extensions of polynomial rings and finite fields. Based on this, we proposed RTL architectures for sequential (SRA) as well as for parallel (PRA) modular reductions designed for simple hardware. SRA can be used to reduce polynomials of whatever degree, and PRA can reduce whole polynomials in one cycle. This reduction works for any random, even secret, polynomial $g(Z) \in \mathbb{GF}(p^m)[Z]$, $\deg g(Z) = t$. The polynomial $g(Z)$ is a variable parameter and can be changed whenever and without any changes of the hardware. Moreover, if the operations in the base field are implemented based on log-and-anti-log and Zech's log techniques, also p, m, and $m(X)$ can by variable parameters. Based on this work, one can design further extensions of rings of polynomials and finite fields in hardware, embedded, as well as in software platforms, where all reduction polynomials can be variable parameters.

Acknowledgment

Supported in part by NATOs Public Diplomacy Division in the framework of "Science for Peace", SPS Project 98452.

References

[1] PATTERSON, N. J.: The Algebraic Decoding of Goppa Codes, IEEE Trans. on Information Theory **21** No. 2 (2012), 203–207.

[2] FARKASOVA, K.—FARKAS, P.—RAKUS, M.—RUZICKY, E.—SILVA, A.—GAMEIRO, A.: Construction of Error Control Run Length Limited Codes Exploiting Some Parity Matrix Properties, Journal of Electrical Engineering **66** No. 3 (2015), 182–184.

[3] MALI, M.—NOVAK, F.—BIASIZZO, A.: Hardware Implementation of AES Algorithm, Journal of Electrical Engineering **56** No. 9-10 (2005,), 265–269.

[4] RAKUS, M.—FARKAS, P.: Double Error Correcting Codes with Improved Code Rates, Journal of Electrical Engineering **55** No. 3-4 (2004,), 89–94.

[5] EGOROV, S.—MARKARIAN, G.: Error Correction beyond the Conventional Error Bound for Reed-Solomon Codes, Journal of Electrical Engineering **54** No. 11-12 (2003), 305=-310.

[6] RAKUS, M.: Comments on Weight Distribution of some Weighted Sum Codes for Erasure Correction, Journal of Electrical Engineering **53** No. 5-6 (2002), 138–142.

[7] HEYSE, S.—GÜNEYSU, T.: Towards One Cycle per Bit Asymmetric Encryption: Code-Based Cryptography on Reconfigurable Hardware, CHES (2012), 340–355.

[8] SHOUFAN, A.—WINK, T.—MOLTER, H. G.—HUSS, S. A.—KOHNERT, E.: A Novel Cryptoprocessor Architecture for the McEliece Public-Key Cryptosystem, IEEE Trans. Computers **59** No. 11 (2010), 1533–1546.

[9] BERNSTEIN, D. J.—LANGE, T.—PETERS, C.: Wild McEliece Incognito, 4th International Workshop, PQCrypto 2011, Proceedings, 2011, pp. 244–254.

[10] REPKA, M.: McEliece PKC Calculator, Journal of Electrical Engineering **65,** No. 6 (2014), http://iris.elf.stuba.sk/JEEEC /data /pdf/6_114-03.pdf.

[11] REPKA, M.—CAYREL, P.-L.: Cryptography Based on Error Correcting Codes: A Survey, Multidisciplinary Perspectives in Cryptology and Information Security (Sattar B. Sadkhan Al Maliky, and Nidaa A. Abaas, ed.), IGI Global, 2014, pp. 133-156.

[12] AN, H.-K.: Fast and Low cost $GF(2^8)$ Multiplier Design based on Double Subfield Transformation, International Journal of Software Engineering and Its Applications **7** No. 4 (2013), 285–294.

[13] CHUANPENG, CH.—ZHONGPING, Q.: Fast Algorithm and Hardware Architecture for Modular Inversion in GF(p), Intelligent Networks and Intelligent Systems, 2009. ICINIS '09. Second International Conference on, 43-45, DOI: 10.1109/ICINIS.2009.20.

[14] MELIKOVIC, N. Z.—STANKOVIC, V.—MILIC, M. L.: Modular Design of Fast Leading Zeros Counting Circuit, Journal of Electrical Engineering **66** No. 6 (2015), 329–333.

Received 8 October 2015

Marek Repka was born in Czech Republic in 1985. Since 2005, he has been studying at the Institute of Computer Science and Mathematics - FEI STU in Bratislava, Slovak Republic. He achieved master degree in the applied informatics focused on security of information systems in 2010. In 2015, he received his PhD degree as a part-time student at the same institute and field, specializing on side-channel-analyses of cryptosystems. He is also interested in application security, and implementation and integration of security controls as well as designing of information security architecture.