Original Article

PERFORMANCE-DRIVEN APPROACHES TO EXPONENTIATION IN GF (2^n)

Alexei Dmitrievich Morozov
Department of Applied Mathematics, Moscow Institute of Physics and Technology, Russia
DOI:https://doi.org/10.5281/zenodo.16409158

ABSTRACT Algebraic operations in Galois fields present properties that render them suitable for use in implementations of cryptographic primitives. Two fundamental operations of interest are modulo squaring and multiplication, whose implementations can be accelerated by using Galois field algebra. An approach is proposed for the acceleration of the calculation of modulo exponentiation in Galois fields, an operation that is fundamental for a wide spectrum of cryptographic algorithms. The approach is based on two developed procedures, namely fast exponentiation to the square and multiplication with a constant number in Galois fields. The proposed innovative accelerated calculation is attained via the use of the properties of the second order polynomial, the Montgomery group reduction and the derivation of pre-calculated tabular results. The mathematical foundation of the proposed method is given, followed by numerical examples that illustrate its operation. The amount of memory required is also calculated. It has been proved, both theoretically and experimentally that the proposed approach renders possible the acceleration of exponentiation in Galois fields by 5 to 7 times, in comparison with known methods.

KEYWORDS multiplication operation on Galois fields, cryptographic algorithms based on Galois Fields algebra, Galois Fields exponentiation, Montgomery reduction, pre computation.

INTRODUCTION operations on Galois fields are performed without the need for operations in Galois fields are widely used in several carry bits, which implies that each bit can be processed in applications of current information technology. One of the parallel. In addition, this leads to specific properties of squaring Transmitted regarding the status of a remote object [7]. Existing DSA-type digital signature mechanisms presuppose the ability to use modular exponentiation on large size numbers. Performing this operation at 4096 most important areas of application of the Galois fields are cryptographic mechanisms for the protection of information. The Galois fields are actively used in stream cipher algorithms [1], can provide the implementation of the fundamental crypto graphic transformations of the AES algorithm [2, 3], form the basis for the creation of cryptographic mechanisms of public key algorithms, such as digital signatures [4, 5] and are used for the zero-knowledge authentication of

Original Article

remote users [6, 7]. For cryptographic applications, the attractive aspect of using the Galois field's algebra is that by carefully selecting the polynomial to define the field, it becomes possible to generate a set of algebraic bases, for which the results of the operations are different. This opens up the possibility of significantly increasing the security level without increasing the bit length of the numbers processed. However, the principle attractive feature for using Galois field algebra to implement public key cryptography, is that they enable an increase in the speed of computer implementation of the corresponding information security protocols by an order of magnitude. This is due to the fact that on Galois fields [5], which can be used to accelerate this operation many times over. The above allow the consideration of public key cryptographic mechanisms based on Galois field algebra as an effective alternative to classical algorithms based on modular arithmetic. Such algorithms are of particular interest for low power terminal microcontrollers of remote-control systems used in IoT technologies. This class of computational platforms is continuously expanding due to the rapid improvement of Internet technologies, expansion of connectivity and availability of inexpensive radio communication equipment. Using the Internet as a medium for data exchange with remote control systems of real world objects, provides many advantages. It is however also associated with a significant number of problems. The most important of these problems is the necessity to ensure the protection of data transmitted over the potentially open Internet [6]. For remote control systems, the most prominent threat is external intervention in their operation. For the protection against such intervention, it is necessary to utilize digital signature technologies for each control message to the terminal microcontroller or data item bits resolution requires a significant amount of computational effort. This is a particularly important obstacle when implemented on a low power microcontroller, which is utilized for implementing real-time control of remote equipment. One of the available ways of overcoming this situation is the use of Galois field algebra combined with research of techniques for the acceleration of exponentiation of large numbers. The purpose of this research is the acceleration of the operation of exponentiation in Galois fields that can provide the foundation for a large number of cryptographic protocols.

STATE OF THE ART: PROBLEM STATEMENT AND REVIEW OF CURRENT TECHNIQUES

Interest in the practical use of the of finite Galois fields $GF(2^n)$ algebra for creating fast mechanisms of public-key cryptography as an alternative to classical technologies based on modular arithmetic, stimulates research for creating efficient methods of the fundamental operation of exponentiation on Galois fields [8, 9].

When using the Galois field algebra $GF(2^n)$, a polynomial representation of numbers in the form

A(x) □ an□1xn□1 □ an□2xn□2 □...□ a1x □ a0 (1)

□j□{0,1,...,n □1}:a_j □{0,1} is commonly used, which corresponds to the usual representation of the number

A □ an□12n□1 □ an□22n□2 □...□ a12 □ a 0 (2)

Addition in Galois Fields is preformed via the logical XOR operation and will be from here on represented by the symbol '□' [6]. Reduction or the calculation of the remainder of a polynomial division A(x) by the produced polynomial P(x) in the Galois field will be represented as A rem P, so as to be distinguishable from the division of number A by number M in regular algebra, denoted as A mod M. The multiplication operation in the Galois fields A □ B rem P, consists of two operations: polynomial multiplication, denoted by '□' and the reduction of the polynomial produced by the field polynomial P [10].

The operation of calculating the square of a number A in the Galois field with the produced polynomial P is denoted as A□A rem P or A|2 rem P. Consequently, the exponentiation operation in the Galois fields, i.e. the

Original Article

calculation of the remainder of the polynomial division of the number A raised to the power E in the polynomial P is denoted as A|E rem P [10]. The existing exponentiation technologies, both in traditional algebra and in Galois fields are based on a classical algorithm that performs a bit-by-bit analysis of the bits of the exponential code

$E = \{e_{n-1}, e_{n-2}, e_0\}, \square j \square \{0, 1, n-1\}; e_i \square \{0, 1\}$ (3)

At every step, a calculation of a square in the Galois field is performed, and the multiplication operation is dependent on the value of the current bit of the exponent. Given that in every next step, the results of the previous step are used, the algorithm cannot be parallelized at the level of the processing of the bits of the exponent.

There exist two variants of this algorithm that differ as to the direction of the analysis of the bits of the exponent. When the exponentiation starts at the most significant bits of the exponent, in each of the n steps the calculation of the square of the current result is performed and the result is multiplied by A if the current bit of the exponential code is 1. Therefore, the mean time t_0 for the exponentiation starting from the most significant bits is $1.5 \square n \square t_m$, where t_m is the time required for the multiplication in the Galois field. The advantage of the exponentiation starting from the least significant bit is that there exists a possibility for partial parallelization of the calculations in one step. This enables the acceleration of the calculations by 1.5 times [10]. Further acceleration of the exponentiation in the Galois field may be attained by reducing the times required for the multiplication and for calculation of the square. In turn, the multiplication and the power calculation in the Galois fields consists of operations performed on large n bit numbers and on small numbers whose bit size is smaller than that of the processor. In actual cryptographic systems that use exponentiation in Galois fields the value of n is 2048 or 4096 i.e. 1 – 2 orders of magnitude larger than that of microprocessors (8 – 64). Hence, for the comparative evaluation of efficiency, it is recommended to consider only operations on large numbers. The multiplication operation consists of two phases: the polynomial multiplication and the reduction of the obtained result. The polynomial multiplication of the n bit numbers requires $2 \square n$ shift operations (both operands are shifted) and, on average, $0.5 \square n$ logical additions operations (XOR). All the above operations belong to the logical class and are executed in approximately the same time. It can therefore be assumed that the logical multiplication requires, on average, $2.5 \square n$ logical operations. The polynomial division is also performed in n cycles, each of which consists of a shift of the code of the Galois field production polynomial and of the test code and, with probability 0.5, the logical addition. Hence the multiplication of the Galois field is performed in $2.5 \square n$ logical operations [7].

There are two main approaches used for the acceleration of the multiplication operation in the Galois fields:

1. Application of preliminary calculations for the acceleration of the reduction
2. Combination of the polynomial multiplication and of the reduction

Most of the existing methods [10, 11] that use the first approach use pre-calculations depending on the produced polynomial P(x) of the Galois field, based on the fact that in real cryptographic protocols, this is part of the public key and can consequently be considered as constant [12]. It may therefore be pre-calculated and be stored for n remainders.

$Q_{n-1} = 2^{2 \square n-1}$ rem P(x), $Q_{n-2} = 2^{2 \square n-2}$ rem P(x),…,

$Q_1 = 2^{n+1}$ rem P(x), $Q_0 = 2^n$ rem P(x) (4)

After the division of the produced polynomial P(x) of the Galois field the reduction consists of the calculation of the logical sum of the values of the table the correspond to the ones in the most significant bits of the produced polynomial. Due to the pre-calculation, the number of logical operations is reduced to $1.5 \square n$. If there exist memory resources for the storage of a table of $n \square 2^h$ values, during the reduction operation, h bits of the product

Original Article

may be processed in parallel. Therefore, the time required for the performance of the reduction operation becomes smaller by h times [13]. The combination of the polynomial multiplication and reduction was proposed in [14]. The described method is based on solving a sequence of congruence's that are derived from the Grobner theory bases in modules over the polynomial ring GF (p). Such a solution is effective for implementation in hardware, when it is feasible to perform the parallel implementation of the logical sum of multiple numbers. Hardware implementations enable an increase in the speed and stability of exponent computation on Galois fields by accelerating the squaring on Galois fields [15]. Another option for the combination of polynomial multiplication and reduction that is oriented towards software implementations, is the use of the Montgomery technology that was modified for Galois fields [16-18]. According to this approach, the combined operation is performed in n steps, each one of which involves the processing of the least significant bit of the multiplier and the logical addition with the binary code of the result, if this bit is equal to 1. Additionally, depending on the least significant bit of the result, the logical addition with the producing polynomial of the field is performed. The result and the bit code of the multiplier are then shifted. As a result, the average number of logical operations is 3□n [19]. Hence the total number T_0 of logical operations performed in large n-bit numbers for the performance of the exponentiation in Galois fields using multiplication for the calculation of the square in the context of the Montgomery combination of multiplication and reduction is given by the formula [20]:

T0 □ n□(3□n □ 0.5□3□n) □ 4.5□n 2 (5)

In [20], a variant of the combined use of pre-calculations depending on the constitutive polynomial of a Galois field and Montgomery reduction for accelerating the squaring is proposed. However, these studies lack an effective implementation of the above idea into another component - multiplication by a constant number and into the exponentiation process on Galois fields in general. An analysis of the algorithm of classical exponentiation in Galois fields demonstrates that 2/3 of the volume of computations are devoted to the calculation of the square [21]. Consequently, the most promising direction for the increase of the speed of exponentiation in Galois fields, is research for the reduction of the computational complexity of the operation of the calculation of the square.

NOTABLE PREVIOUS RELATED WORK

This problem has attracted significant attention from researchers, due to its importance for providing enhanced information security [22]. Additionally, it has attracted attention due to the possibility of applying accelerated calculations in other fields as well. In [23], modular arithmetic used for error correction coding could benefit from the application of this technique. Accelerated modular arithmetic is also applied in the case of wireless sensor networks [24]. Other researchers have investigated the possibility of employing accelerated calculations in hybrid random number generators [25] and emphasizing in serial encryption [26]. Accelerated modulo arithmetic operations have also been recognized as capable of also benefiting cryptographic hardware implementations [27].

STATE – OF – THE ART IN CURRENT RELATED TECHNIQUES

The proposals by Zhang [28] involve an improved Barrett modular multiplication (BMM) algorithm along with a hardware-efficient design. The key idea is to parallelize the quotient estimation and intermediate product computations, and to replace costly multi-word additions with lightweight carry-save compression operations. They introduce a novel data representation allowing use of tiny (2-bit and 3-bit) adders for certain overflow and partial-sum corrections. In an FPGA implementation, their optimized Barrett multiplier significantly outperforms both classical Barrett and Montgomery multipliers in terms of speed and area, especially for high-radix (large word size) arithmetic. This demonstrates that even a well-known method like Barrett's can be tweaked at the

Original Article

algorithmic level (quotient estimation and reduction steps) for notable efficiency gains in cryptographic hardware. In [29], the researchers present a software-side innovation that benefits modern CPUs with 512-bit SIMD instructions. They focus on the 52-bit fused multiply-add (VPMADD52) capability of the Intel AVX-512 to perform batch Montgomery multiplications in parallel. A novel contribution is their Truncated Montgomery Multiplication, which computes only the necessary lower half of certain intermediate products, reducing workload in the reduction phase. This optimization yields ~20% speedup in the inner multiplication loop compared to conventional Montgomery multiplication. By processing up to 8 modular multiplications in a word-sliced SIMD batch, they achieved over 4× faster modular multiplication throughput than GMP and OpenSSL for operands of 1024 up to 4096-bit. For full modular exponentiation, their 512-bit wide implementation attained 1.75× speedup for 1024-bit exponents (and 1.38× for 2048-bit) over OpenSSL's AVX2-based constant time exponentiation routine. The work in [30] focuses on the scenario where the base a is fixed across many exponentiations (common in certain protocols or repeated operations). They implemented a Montgomery-arithmetic based exponentiation in C++ and introduced a pre computation of a reduced residue table for powers of the fixed base. Using a right-to-left binary exponentiation with that pre computed table, they achieved notable speedups for large exponents (larger than 1024 bits). In [31], the proposals tackled the problem of multiple simultaneous exponentiations which occurs in multi-base cryptographic protocols, batch verification, etc. They compared known multi-exponentiation algorithms (such as interleaving exponentiation vs. separate exponentiations) not just by counting multiplications, but by actual execution time on different hardware. A key finding is that the theoretically optimal algorithm (in terms of minimal multiplications) might not be the fastest in practice once factors like memory access and pipeline stalls are considered. Although not primarily about speed, it's worth noting that in cryptographic contexts, sometimes a slightly slower algorithm is chosen to prevent timing or cache side-channels. For instance, a Montgomery ladder or a fixed-window method with dummy operations may be used to make execution time independent of secret exponents. Some recent research [32] evaluated the security of such implementations against cache attacks. While these works focus on security, they often propose minor tweaks that can reduce the performance penalty of constant-time exponentiation.

In the hardware front, researches have demonstrated [33] that RSA and modular exponentiation can be performed entirely in Residue Number System (RNS), eliminating the need for costly base-conversion steps at every multiplication. They leverage a recent technique called Sum of Residues reduction, which performs modular reduction within a single RNS system (as opposed to earlier RNS Montgomery methods that required two related RNS systems and multiple base extensions). By improving both the algorithm and the digital architecture, they achieved a 1024-bit modular exponentiation in only 0.567 ms on a Xilinx FPGA (Virtex-6), using a reasonable amount of resources. Other research [34] has addressed the efficiency of RNS from another angle, by looking at how to choose the RNS moduli for optimal performance and simpler conversion. The proposals of [34] introduced the selection of two new balanced RNS bases that are well-suited for Montgomery multiplication. In particular, they chose moduli that are well-formed, i.e. close to powers of 2, in order to simplify the reduction and base extension operations. They also designed efficient reverse converters to recombine residues that take advantage of this balanced structure. In FPGA experiments, their RNS Montgomery multiplier shows excellent speedup for large operand multiplication, with manageable hardware cost. The research proposed in [35], develops a Block-Parallel approach to exponentiation using Intel AVX-512 and demonstrates its benefit not only for speed but also for fault attack countermeasures. The proposal introduces the Block Product Scanning (BPS) method (a

Original Article

blockbased Montgomery multiply) which vectorizes big-integer ops. On an Intel Xeon, their implementation achieved 1.5× higher RSA throughput than GMP 6.1.2 for 2048-bit exponentiation and 1.9× faster RSA decryption compared to OpenSSL, thanks to AVX-512 parallelism. There is also ongoing work in designing application-specific integrated circuits for tasks like the Verifiable Delay Function (VDF), which essentially requires computing a^{2t} mod N squaring operations [36]. Researchers have in this case [36] proposed pipelined squaring units and even ASIC prototypes that perform a 2048-bit modular squaring in under 100 ns, targeting VDF use-cases. These specialized designs often use high-radix Montgomery multipliers and deep pipelines to churn out one modular square per clock cycle after latency. Resource-constrained devices (IoT sensors, microcontrollers) may not be able to perform 2048-bit exponentiations quickly. One approach highlighted in recent work is the secure outsourcing of modular exponentiation to a more powerful server. Protocols have been proposed for an IoT device to transfer RSA computations to the cloud, for which a vulnerability is shown in [37] and a fix is provided. The improved protocol of [37] ensures that the untrusted server cannot learn the secret exponent or result, by blinding intermediate values, and is resilient against known lattice attacks. The overall performance remains unaffected by this correction and hence the computational effort benefit is achieved with better security. This approach represents a workaround for a cryptographic solution to the performance problem, since it allows small devices to benefit from big accelerators in the cloud. When combined with algorithmic improvements, whereby the cloud server can use all the methods discussed above, secure cloud computations can make heavy cryptography feasible in lightweight environments without sacrificing privacy. Finally, research has also been directed toward algorithmic developments, such as the efficient implementation of Montgomery multiplication and Barrett reduction algorithms. In [38], a high-efficiency digital signal processing framework is explored, aimed at optimizing modulo calculations, emphasizing significant improvements in efficiency through these established methods. By formulating optimizations specifically for lattice-based cryptography, the authors demonstrated the applicability of this approach to postquantum systems.

C. CONTRIBUTIONS OF THE PRESENT WORK

The principal disadvantage of the existing methods for accelerating multiplicative operations on Galois fields is that they do not use the possibilities of simultaneous multiple digit processing. The possibility of using this reserve for increasing the speed of computation is due to both the specific features of operations on Galois fields and the extension of the Montgomery reduction operation. In the following sections, existing techniques will be presented in detail for accelerated calculation of the square of a number and the accelerated calculation of the multiplication with a constant number in Galois fields. Following that, based on these accelerated procedures for squaring and multiplication, an innovative proposed procedure will be developed for the execution of exponentiation in the Galois fields i.e., for the calculation of $A|^{E}$ rem P.

III. ACCELERATED SQUARING METHOD IN GALOIS FIELDS WITH MONTGOMERY GROUP REDUCTION

When using both versions of the classical algorithm for exponentiation on Galois fields of long numbers, the squaring operation takes 2/3 of the volume of all calculations. Therefore, to accelerate exponentiation on Galois fields, it is necessary to investigate the possibilities of reducing the time required for performing squaring operations.

The principal resources that may be used for the reduction of the number of logical operations for the squaring in the Galois fields are:

Original Article

- the Factor Square Property (FSP),
- application of the Montgomery reduction modified for Galois fields,
- Group processing of bits when performing Montgomery reduction.

The property of the calculation of the square of a polynomial of A, is that this operation is actually equivalent to inserting zeros between the bits of the number A. Indeed, the polynomial product is the XOR of the logical products of all possible pairs of bits of the factor codes, multiplied by the corresponding power of two. If the factors are equal to each other, then the logical product of a pair of different bits is included in the XOR twice and, consequently, it is cancelled out. Therefore, the polynomial square can be represented as:

n□1 n□1 n□1

A□ A □ □□ai □a j □2i□ j □ □a j □22□ j (6)

j□0 i□0 j□0

For example, if A=9=1001 $_2$, then A□A=1000001$_2$=65. Accordingly, the implementation of polynomial squaring becomes significantly simpler and faster than calculating the square in traditional algebra. Polynomial squaring can hence be performed in software using 2□n shift operations of the n-bit code A. After calculating the polynomial square, it is necessary to perform its reduction, that is, the calculation of the remainder of the division by the field generator polynomial – P. Direct execution of the polynomial division operation of the 2□n -bit code of the polynomial square by the field generator polynomial requires n shift operations of the n-bit code P and n/2 XOR operations on n-bit codes. Thus, sequential execution of polynomial squaring and reduction of the obtained result requires 3.5□n logical operations on n-bit codes. A more efficient implementation of squaring on Galois fields is achieved by combining the bit expansion of the number a is being squared with the reduction. Such a combination is possible only when performing the reduction from the lowest bits that is, using the Montgomery's technique [5]. In order to attain this, the technique which is used for modular reduction in ordinary algebra, must be updated in order to encompass the features of reduction on finite Galois fields. For the implementation of the combination of polynomial squaring and modified Montgomery reduction on the Galois field, a procedure for fast calculation of A^2 rem P is proposed. In the developed procedure, the code of the n-bit number A, which is raised to the power of E on the finite Galois field GF(2^n), is divided into two fragments: the n/2 least significant digits form the first fragment A_1, and the n/2 most significant digits of the number A form the second fragment A_2. The procedure for the combined polynomial squaring and modified Montgomery reduction on the Galois field involves the following steps:

1. The initial value of the variable R of the current result and the index j of the loop are set to zero: R=0 and j=0.
2. If the index j of the loop is even, that is j mod 2=0, then the least significant digit a_{10} of the code A_1 is logically added to the least significant bit of R, and the least significant bit a_{20} of the code A_2 is logically added to the most significant $(n+1)^{th}$ bit of R=R □ a_{10} □ a_{20}□2^n.
3. If the least significant digit of R is equal to one: r_0 =1, then the generator polynomial of the field P is logically added to P: R = R □ P.
4. The code R is shifted to the right by one bit: R>>=1.
5. If the index j of the loop is even, that is, j mod 2=0, then the codes A_1 and A_2 are shifted to the right: A_1>>=1 and A_2>>1.
6. The index j of the loop is increased by one. If j<n, then a return to repeat Step 2 is performed.

Original Article

7. End of process. The value R = A□A□ Q^{-1} rem P is obtained, where Q^{-1} is the multiplicative inverse of the polynomial of $Q(x)=x^n$ in the Galois field. The polynomial P(x) is formed, i.e. Q□Q^{-1} rem P =1.

In order to obtain the correct value of the squaring of the number A in the Galois field, the result of the above procedure must be multiplied by Q:

R□= R□Q rem P (7)

However, the correction is not performed during the exponentiation.

The proposed procedure for the squaring in the Galois field is illustrated by an example of the squaring of a number

$A = 9_{10} = 1001_2$ (8) In the Galois field, the polynomial

$P(x) = x^4+x+1$ (9)

is formulated, that corresponds to the number

$P=10011_2=19_{10}$; n=4, (10)

And

$Q=10000_2=16$, $Q^{-1} =14_{10} =1110_2$. (11) Indeed,

Q□Q^{-1} rem P = 16□14 rem 19 = 1. (12) The actual result is then,

R□=A□A rem P = 9□9 rem 19=13. (13)

The step-by-step modification of the variables R and A during the execution of the proposed procedure of the calculation of the square in the Galois field for A = 9 with the formulation of the polynomial $P(x) = x^4+x+1$ is presented in Table 1.

The result R is the product A□A□Q^{-1} rem P = 9 □ 9 □ 14 rem 19 = 10. In order to obtain the actual result of the squaring of the number A=9 in the Galois field it is necessary to multiply R with the value

Q: R□ = R □ Q rem P = 10□16 rem 19 = 13. (14)

The dynamic progress of variables R and A during the execution of the calculation of the square of A=9 in the Galois field, from the formulated polynomial $P(x) = x^4+x+1$ is illustrated in Table 1 below.

Table 1. Evolution of the calculation for $A = 9_{10}$

j	Transformation R				Transformation A	
	R	R=R□a_{10}□a_{20}□2^n	R=R□P	R>>=1	$A_2=10_2$	$A_1=01_2$
0	0	R=0□1□ 0=1	00001□10011=10010	01001	01	00
1	10010	-	01001□10011=11010	01101	-	
2	01101	01101□10000 =11101	11101□10011=01110	00111	00	00
3	00111		0111□10011=10100	01010	-	

The execution of the above procedure involves performing n/2 shifts of the two halves A_1 and A_2 of number A, n shifts of number R, and, on average, n/2 logical summation operations (XOR). All these operations are performed on n-bit codes. Given that n is much greater than the processor capacity r: n>>r, each of the operations described actually requires performing n/r processor instructions. The remaining actions required by the proposed procedure, such as R = R □ a_{10} □ a20□ 2^n or testing the R bits, are performed in 1-2 processor operations, i.e. in significantly less time. Thus, the total number of logical operations on n-bit codes required to implement the proposed squaring procedure on the Galois field is 2□n. This is significantly less than the similar figure of 3.5□n for separate execution of polynomial squaring and reduction on the Galois field. One particular characteristic of

Original Article

the proposed method for accelerated squaring on the Galois field, is that the modification of the least significant digit of the intermediate result R occurs in every second cycle (for even values of j) and this modification concerns only one bit. This enables possibilities for implementing group reduction on the Galois field, in which the shift to the right is performed immediately by k digits. In turn, this allows to reduce the time spent on reduction by k times and, thus, significantly speed up squaring on Galois fields. In order to implement this possibility, it is necessary to logically add to the current code R such a linear combination

$L(P) = \square_{k-1}\square 2^{k-1}\square P + \square_{k-2}\square 2^{k-2}\square P+\ldots+\square_1\square 2\square P+\square_0\square P$, (15)$\square$ $\square i\square\{0,1,\ldots,k-1\}$: $\square_i\square\{0,1\}$, which ensures that the k least significant bits of the logical sum $R\square L(P)$ are equal to zero. It will be shown that such a linear combination L (P) always exists for any value of the code

$R = r_n\square 2^n + r_{n-1}\square 2^{n-1}+\ldots+r_{k-1}\square 2^{k-1}+\ldots+ r_1\square 2 + r_0$, (15) where $\square j\square\{0, 1,\ldots,n\}$: $r_j\square\{0,1\}$, provided that the generating polynomial P(x) is prime. Since P(x), the polynomial generated in the Galois field is prime, then it necessarily contains a non-zero component at x^0 (otherwise, it would necessarily be divisible by the polynomial V(x) = x, i.e., it would not be prime). This implies that the number P that corresponds to the generator polynomial P(x) is odd, i.e., its least significant bit p_0 is equal to one: $p_0=1$. If the least significant bit r_0 of the code R being reduced is equal to one, then its logical sum with P in the least significant bit is zero. Thus, in order for the logical sum $R\square L(P)$ to have zero in the least significant bit, it is necessary that $\square_0 = r_0$. Similarly, if the second bit of the logical sum $R\square\square_0\square P$ is equal to one, then for $\square_1=1$ the bit of the logical sum $R\square\square_0\square P\square\square_1\square P\square 2$ with the same name is equal to zero. This means that it is always possible to make the two least significant bits of $R\square\ \square_0\square P\ \square\ \square_1\square\ P\ \square\ 2$ equal to zero: for this it is necessary that $\square_1 = r_1\ \square\ \square_0\ \square\ p_1$. Reasoning in a similar manner, it is easy to show that in order for the three least significant bits of the logical sum $R\square\square_0\square P\square\square_1\square P\square 2\square\square_2\square P\square 2^2$ to be equal to zero, it is sufficient to satisfy the condition $\square_2=r_1\square\square_0\square p_1\square\square_1\square p_2$.

Continuing the above reasoning, we can come to the conclusion that it is always possible to choose binary coefficients $\square_0, \square_1, \ldots, \square_{k-1}$ in such a way that the k least significant bits of the logical sum

$R\ \square\ \square_0\square P\ \square\ \square_1\square P\ \square\ 2\ \square\ \square_2\square P\ \square 2^2\square\ \ldots\ \square\ \square_{k-1}\square P\ \square\ 2^{k-1}$ (16) are equal to zero. The proof of the above statement enables the organization of a simultaneous Montgomery reduction immediately over k bits of the current result when squaring on Galois fields. Subsequently, due to this, it hence becomes possible to significantly speed up the fundamental operation of exponentiation on Galois fields. For this reason, it is proposed that, for a given polynomial P(x) formulated in the Galois field and for given values $r_{k-1}, r_k2, \ldots, r_1, r_0$, it is possible to obtain as a result the corresponding values $\square_{k-1}, \square_{k-2}, \ldots, \square_1, \square_0$ with a recursive application of the approach outlined above. Hence, for each of the possible of the 2^k-1 combinations (excluding zeros) of the k bits of the code $r_k1, r_{k-2}, \ldots, r_1, r_0$, the values of the sums $L(P)=\square_{k-1}\square 2^{k-1}\square P + \square_k2\square 2^{k-2}\square P+\ldots+\square_1\square 2\square P+\square_0\square P$ are derived, for which the k least significant bits of L(P) are equal to the corresponding combination. The results of the calculation are presented in the form of the 2^k-1 table of values T(1), T(2),…,T(2^k-1). The value of k is selected to be even and such that n is exactly divisible by k. The above method is illustrated according to the following example. Let n=8 and the Galois field formulated by the polynomial $P(x)=x^8+x^7+x^6+x^5+x^3+x^2+1$. For n=8, the number $Q=2^n=256$ and the multiplicative inverse Q^{-1} that is produced with the given polynomial P(x) is $Q^{-1}=127$. Then indeed, $256\square 127$ rem P(x)=1. This polynomial corresponds to the number $P=111011101_2 = 477_{10}$. The four least significant digits for this number (for k=4) of this number are: $p_0=1$, $p_1=0$, $p_2=1$ και $p_3=1$.

For each of the 16 possible values of $r_3\square 8+r_2\square 4+r_1\square 2+r_0$, the values of the coefficients $\square_0$, $\square_1$, $\square_2$ and $\square_3$ of the linear combination L(P) can be calculated based on the above considerations. The values of the linear

Original Article

combinations calculated in this way are given in Table 2. This table summarizes values of the pre-calculated results for the Galois field with polynomial $P(x)=x^8+x^7+x^6+x^5+x^3+x^2+1$ for k=4.

Additionally, in order to quickly formulate the k bits of a polynomial for the calculation of the square of k/2-bit packets of a number via the insertion of zeros between the digits of the binary representation, the formulation of table Z is proposed. This table contains in each case, polynomials with squares obtained by the insertion of for each one of the $2^{k/2}$-1 codes of k/2-bits. Specifically for, k=4 the table Z consists of three rows: $Z[1]=Z[01_2]=0001_2$, $Z[10_2]=0100_2$ and $Z[11_2]=0101$. These values may be identified as a subset of the rows of Table 2.

All the steps described above, depending on the polynomial P(x) that has been formulated and the number k of the concurrently processed bits, need to be performed once for actual cryptographic data protection systems, since the polynomial is part of the public key.

Table 2. Pre-calculated results for P(x) and k=4

r3r2r1r0	T	r3r2r1r0	T
		1000(8)	3816_{10} =1110 1110 1000_2
0001(1)	2113_{10}=1000 0100 0001_2	1001(9)	1705_{10} = 0110 1010 1001_2
0010(2)	3410_{10}=1101 0101 0010_2	1010(10)	954_{10} = 0011 1011 1010_2
0011(3)	1299_{10}=0101 0001 0011_2	1011(11)	3067_{10} =1011 1111 1011_2
0100(4)	1908_{10}=0111 0111 0100_2	1100(12)	2460_{10} =1001 1001 1100_2
0101(5)	3893_{10}=1111 0011 0101_2	1101(13)	477_{10}= 0001 1101 1101_2
0110(6)	2598_{10}=1010 0010 0110_2	1110(14)	1230_{10} = 0100 1100 1110_2
0111(7)	615_{10}=0010 0110 0111_2	1111(15)	3215_{10} = 1100 1000 1111_2

It is proposed to calculate the square A□A rem P of the number A in the Galois field according to the following sequence:

1. The cycle count j is initialized: j=1. The code of the result is also initialized (n + k)- bit R: R=0.
2. The value of R is shifted by k bits: R>>=k. The most significant k bits of R are assigned values from the table, the index of which is determined by the least significant k/2 bits of A: $Z(a_{k/2-1}, a_{k/2-2}, \ldots, a_1, a_0)$.
3. If the least significant k bits R: $r_{k-1}, r_{k-2}, \ldots, r_0$ are equal to zero – go to Step 5. Otherwise R is logically added upon the code $T[r_{k-1}, r_{k-2}, \ldots, r_0]$: R = R □ $T[r_{k-1}, r_{k-2}, \ldots, r_0]$.
4. A shift of A is performed by k/2 bits: A>>=k/2. Increment the counter j: j=j+1. If j □ 2□n/k, then return to Step 2.
5. End of process. The value R = A □ A □ Q^{-1} rem P is obtained

The proposed procedure for the accelerated squaring in the Galois field is illustrated using the following example. Consider squaring the number $A=159_{10}$ = 1001 1111_2 in the Galois field with the forming polynomial $P(x)=x^8+x^7+x^6+x^5+x^3+x^2+1$ for which and for k=4, Table 2 is constructed. The true value of the result A□A rem P = 159 □ 159 rem 477 = 1110111_2 = 231.

The dynamic progress of R and q in the steps j of the proposed procedure for squaring A□A rem P for A = 159 and P = 477 for k = 4, is shown in Table 3.

Original Article

The result R=236 is different from the correct result and is the product A□A□Q^{-1} rem P = 159□159□127 rem 477. In order to obtain the correct result R□ for the number A=159 in the Galois field, it is necessary to perform the Montgomery correction, that is to multiply the obtained result R by the value of Q: R□=R□Q rem P = 236□256 rem 477 = 231.

Table 3. A□A rem P for A = 159, P = 477, k = 4.

j	Operations on R		Operations on A A>>=2
	Logical Addition (XOR)	Shift (R>>=4)	
0	0000 0000	0000 0000 0000	1001 1111
1	-	0101 0000 0000	0010 0111
2	-	0101 0101 0000	0000 1001
3	-	0001 0101 0101= 341	0000 0010
4	R=R□T[5] =341□3893 = 3680=1110 0110 0000	0100 1110 0110 = 1254	0000 0000
5	R=R□T[6] = 1254□2598= 3776= 1110 1100 0000	0000 1110 1100 = 236	

During the exponentiation in the Galois field in information security systems, the actual word length n (typical values of which are 2048 or 4096) of the operands is one to two orders of magnitude higher than the bit capacity of the processor. Consequently, for the estimation of the number of operations required for the squaring, one can ignore the operations concerning operands the size of which is smaller than the capacity of the processor and consider only operations on long operands i.e., operands for n bit operations. The performance of the above procedure involves the execution of n/k shifts of the number A, 2□n/k 2 shifts of the number R and n/k logical additions (XOR). Hence the total of logical operations required for the application of the proposed procedure of the squaring in the Galois field is 4□n/k. This implies that the use of the use of the reduction of the Montgomery group with the concurrent k bit processing, renders feasible the acceleration of the squaring in the Galois fields by a factor of 0.75□k times.

IV. ACCELERATED MULTIPLICATION BY A CONSTANT NUMBER ON GALOIS FIELDS WITH MONTGOMERY GROUP REDUCTION

When using the Montgomery reduction as modified for the Galois fields, one may use the accelerated multiplication in the

Galois fields i.e., the calculation A□B rem P, where

$A=a_{n-1}\square 2^{n-1}+a_{n-2}\square 2^{n-2}+\ldots+a_2\square 2^2+a_1\square 2+a_0$, $B=b_{n-1}\square 2^{n-1}+b_{n-2}\square 2^{n-2}+\ldots+b_2\square 2^2+b_1\square 2+b_0$, (17)□

□i□{0,1,…,n-1}: a_i,b_i□{0,1}.

Similarly to the proposed method for fast squaring, multiplication in the Galois fields may be accelerated via:

- The application of the Montgomery reduction as modified for the Galois fields
- Concurrent processing of digits during the execution of the Montgomery reduction.

For the purpose of immediately reducing the q least significant bits of the intermediate result using the Montgomery technology during the calculation of the product, it is recommended to use the pre-calculation tables.

Original Article

The logical addition of the values of the table to the intermediate result, allows the assignment of zeros to its q least significant bits. However, in contrast to the squaring, these values depend not only on the produced polynomial P(x) of the Galois field, but also on the multiplier A. It is therefore necessary to perform preliminary multiplications before every calculation of the exponent $A|^E$ rem P.

In every step of the multiplication for the intermediate result R, a logical addition of the code $Y = b_{q-1} \square 2^{q-1} \square A + b_{q-2} \square 2^{q-2} \square A + \ldots + b_1 \square 2 \square A + b_0 \square A$ is performed, that depends on the q least significant bits $b_{q-1}, b_{q-2}, \ldots, b_0$ of the multiplier B and the multiplicand A. The compensatory code $D = v_{q-1} \square 2^{q-1} \square P + v_{q-2} \square 2^{q-2} \square P + \ldots + v_1 \square 2 \square P + v_0 \square P$ of the Montgomery reduction has to be added upon the logical sum $R \square Y$, that is selected so that the q least significant bits of the sum $R \square Y \square D$ are equal to zero. In the previous section, it was shown that the compensatory code D exists for any value q of the least significant bits of the logical sum $R \square Y$. From this it follows that for given values of $r_{q-1}, r_{q-2}, \ldots, r_1, r_0$ of the q least significant bits of the intermediate result R and of the q least significant bits $b_{q-1}, b_{q-2}, \ldots, b_0$ of the multiplier B, there always exists such a set $v_{q-1}, v_{q-2}, \ldots, v_0$, for which the q least significant bits of the logical sum $R \square Y \square D$ are equal to zero. Consequently, the numbers of the rows of the table W of the pre-calculated values are determined by the given codes $r_{q-1}, r_q 2, \ldots, r_1, r_0$ and $b_{q-1}, b_{q-2}, \ldots, b_0$ and the values of the table contain the code $D \square Y$, For which the q least significant bits of the logical sum $R \square Y \square D$ are equal to zero. Hence the value of the $2 \square q$ bits of the j [th] line of the table are formulated as:

$$j \square \square r_i \square 2^{q \square i} \square \square b_l \square 2^l \quad (18)$$

$i \square 0 \qquad l \square 0$

For example, for n = 6, q = 2, $A = 57_{10} = 111001_2$ and the polynomial formulated in the Galois field is $P(x) = x^6 + x^4 + x^2 + x + 1$ (19) which corresponds to the number $P = 87_{10} = 1010111_2$. The precalculated values of $D \square Y$ for all the possible values of the 2-bit codes r_1, r_0 and b_1, b_0 for A = 57 and P = 87 are presented in Table 4 below. On algorithmic level and the general case, this table will be referred to as table W.

Table 4. Pre-calculated values for A = 57 and P = 87

r_1,r_0	b1,b0	j	D	v_1,v_0	r_1,r_0	b_1,b_0	j	D	v_1,v_0
0 0	0 0	0	0	0 0	1 0	0 0	8	174_{10} = $1010\ 1110_2$	1 0
0 0	0 1	1	192_{10} = $1100\ 0000_2$	1 1	1 0	0 1	9	110_{10} = $0110\ 1110_2$	0 1
0 0	1 0	2	220_{10} = $1101\ 1100_2$	1 0	1 0	1 0	10	114_{10} = $0111\ 0010_2$	0 0
0 0	1 1	3	28_{10} = $0001\ 1100_2$	0 1	1 0	1 1	11	178_{10} = $1011\ 0010_2$	1 1
0 1	0 0	4	57_{10} = $0011\ 1001_2$	1 1	1 1	0 0	12	87_{10} = $0101\ 0111_2$	0 1
0 1	0 1	5	37_{10} = $0010\ 0101_2$	0 1	1 1	0 1	13	151_{10} = $1001\ 0111_2$	1 0

Original Article

0 1	1 0	6	229_{10} = 1110 0101_2	0 0	1 1	1 0	14	139_{10} = 1000 1011_2	1 1
0 1	1 1	7	249_{10} = 1111 1001_2	1 1	1 1	1 1	15	75_{10} = 0100 1011_2	0 0

The procedure required is the following:

1. The cycle counter i is initialized as i=1;

Similarly, the (n+k)-bit result code R: R =0.

2. For the values of the q least significant bits of R and the q least significant bits B using Equation (18) the corresponding line number j within the pre-calculated table W is determined.
3. The value W[j] of the logical addition is read from the table and hence R: R = R □ W[j].
4. The values R and B are shifted by q bits: R>>=q. B>>=q.

The cycle counter is incremented i: i=i+1.

If i□n/q, return to Step 2.

The operation of the described multiplication procedure with concurrent processing (q=2) in the Galois fields is illustrated via the example of the multiplication A=57 by B=41. The polynomial

$P(x)=x^6 + x^4 + x^2+x+1$ (20) is created. For this particular Galois field

$P = 87$, $Q = 2^6 = 64$, and $Q^{-1} = 9$. (21) The correct value of the product is obtained as 57 □ 41 rem 87 = 18. The calculations of R and A during the steps for all i of the procedure described for the accelerated multiplication in the Galois fields is given in Table 5. This table illustrates in stepby-step form, the evolution of the values of R and A for each iteration of the execution of the calculation of A□B rem P for A = 57, $B = 41_{10} = 101001_2$ and P = 87 for q = 2

Table 5. Iterations for A = 57 and P = 87

i	r_1 r_0	b_1 b_0	j	W[j]	Operations in R		Shift B
					R = R □ W[j]	R >>= 2	B >>= 2
0	0 0	0 1	1	192	0 □ 192 = 192_{10} = 1100 0000_2	11 0000_2= 48_{10}	1010
1	0 0	1 0	2	220	48 □ 220 = 236_{10} = 1000 1100_2	11 1011_2= 59_{10}	0010
2	1 1	1 0	14	139	59 □ 139 = 176_{10} = 1011 0000_2	10 1100_2= 44_{10}	0000

The obtained result R = 44 is different from the true result A □ B □ Q^{-1} rem P = 57 □ 41□ 9 rem 87. In order to obtain the result R□ it is necessary to perform the Montgomery correction i.e., to multiply the result R by the value Q:

R□=R □ Q rem P = 44 □ 64 rem 87 = 18 (22)

The immediate execution of one cycle of the described procedure requires one logical addition and two shift operations. Hence the total number of operations for the application of the procedure for multiplication is 3□n/q.

V. ACCELERATED EXPONENTIATION IN GALOIS FIELDS Based on the accelerated procedures for squaring and multiplication that were developed in the previous sections, the following innovative procedure is proposed for the execution of exponentiation in the Galois fields i.e., for the calculation of $A|^E$ rem P.

Original Article

The creation of the polynomial P(x) for most cryptographic data protection mechanisms based on Galois fields, is part of the public key and changes very rarely. This implies that the table T of the pre-calculations is predetermined, may be stored in memory and the time required to complete it is not considered a computational complexity during the exponentiation of a particular number A.

Before performing the exponentiation, the initial values of the result

$R=Q \text{ rem } P= x^n \text{ rem } P$ (23)

And the parameter

$C = Q \square Q \text{ rem } P = x^{2\square n} \text{ rem } P$ (24) are also calculated. These values only depend on the produced polynomial of the Galois field and need to be once together with the table T of pre - calculations.

The immediate procedure for the calculation of $A|^E$ rem P in the proposed scheme, starts with the formulation of the table W of pre – calculations, that is used for the accelerated multiplication.

The size of this table is $2^{2\square q}$ - 1. For each of the possible values $b_{q-1}, b_{q-2}, \ldots, b_0$ the values

$Y = b_{q-1}\square 2^{q-1}\square A + b_{q-2}\square 2^{q-2} \square A + \ldots + b_1\square 2\square A + b_0\square A$ (25) are calculated, with the operations required being q-1 shifts and $0.5\square q^2$ logical additions.

For each of the possible values $r_{q-1}, r_{q-2}, \ldots, r_1, r_0$, a linear system of Boolean equations is solved, that renders possible the determination of the values $v_{q-1}, v_{q-2}, \ldots, v_0$ for which the q least significant bits of the sum $R \square Y \square D$ are equal to zero. The mean number of operations for the determination of a value v is $0.5\square q$. Consequently, the total number of logical additions required for the determination of the values $v_{q-1}, v_{q-2}, \ldots, v_0$ is $0.5\square q^2$.

Hence the construction of the table W of the pre-Calculations, requires $2^q \square (q + 0.5 \square q^2)$ logical operations for the determination of all possible Y and $2^{q-1}\square q^2$ logical operations for the determination of $v_{q-1}, v_{q-2}, \ldots, v_0$. Taking into account the function for the formulation of $W[j] = Y\square D$ for all rows of the table, the total number T_W of logical operations for constructing this is table is determined by the formula:

$T_W \square 2^{2\square q} \square(q \square 0.5\square q^2) \square 2^q \square(q \square 0.5\square q^2) \square$

$\square (q \square 0.5\square q2)\square(22\square q \square 2q)$ (26)

For the description of the proposed procedure for the fast exponentiation in Galois fields, SM(A,k) denotes the expanded procedure for the fast squaring of a number A in a Galois field with Montgomery reduction group k bits, that formulates $A\square A\square Q^{-1}$rem P. Similarly, MM(A, B, q) denotes the expanded procedure for accelerated multiplication in the Galois field of the numbers A and B with the (q-bit) Montgomery reduction group $A \square B \square Q^{-1}$ rem P. The algorithm can then be described as follows:

For the selected q and the given A, a table W of preliminary calculations is formulated.

1. Calculate G = MM (A,C,q).
2. The number i of the current bit of the binary code of the exponent is set to n: i = n.
3. Raise to the square of the current result R in the Galois field: R = SM(R, k).
4. If the i^{th} bit e_i of the binary code of the exponent is equal to 1 e_i =1 : the current result R is multiplied by G using the procedure MM: R=MM(R, G, q).
5. If i > 0 the counter is decremented (i = i - 1) and the process returns to Step 4.
6. The correct result R□ is formulated as the multiplication of

R with unity: R□ = MM(R, 1,q)

Original Article

The proposed procedure is illustrated via an example of the exponentiation of the number A = 159_{10} to the exponent E =203_{10} = 1100 1011_2 in the Galois field, that is formulated by the polynomial P(x) = $x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + 1$, corresponding to the number P = 477; n = 8. The correct result is $159|^{203}$ rem 477 = 69.

For a single repetition and for a given constant polynomial P(x) the calculations R = Q rem P = x^n rem P = 256 rem 477 = 221 and C = Q □ Q rem P = $x^{2\square n}$ rem P = 2^{16} rem 477 = 97 are performed. For the selected value of k, the tables T of preliminary calculation are created and stored.

The process of the exponentiation begins immediately with the formulation of the table W of the pre-calculations (Step 1) and the calculation G=MM(159, 97, q) = 105 (Step 2).

The dynamic evolution of the values of the current result during the execution of Steps 4 – 6 of the proposed procedure are presented in Table 6. This table illustrates the values of the pre-calculated results for the Galois field with polynomial P(x) = $x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + 1$ for k = 4.

According to Step 7 of the procedure, the obtained result R = 124 is corrected by multiplying with unity: R□ = MM(R, 1, q) = MM(124, 1, q) = 69.

It is apparent that for each of the n iterations of the described procedure for multiplication in the Galois fields a squaring with 4 □ n / k logical operations is required, and with probability 0.5, an additional multiplication with 3□n/q logical operations is also required. Additionally, the formulation of the table W is performed before the repetitions that requires T_W logical operations.

Table 6. Steps 4-6 for A = 159_{10} and E = 203_{10}

i	e_i	Evolution of R	
		Squaring	Multiplication in G
8	1	SM(221,k) = 221	MM(221,105,q) = 105
7	1	SM(105,k) = 28	MM(28,105,q) = 65
6	0	SM(65,k) = 111	
5	0	SM(111,k) = 185	
4	1	SM(185,k) = 77	MM(77,105,q) = 223
3	0	SM(223,k) = 252	
2	1	SM(252,k) = 166	MM(166,105,q) = 250
1	1	SM(250,k) = 3	MM(3,105,q) = 124

Hence the total number of logical operations T_E required for the exponentiation in the Galois fields according to the proposed method is given by the formula:

$$T_E \square T_W \square n \square \left(\frac{2}{2\square q} \square \frac{4}{q} \square \frac{3}{2} \square 0.5 \square \frac{1.5}{2} \right) \square k \quad q$$

$$\square (2 \quad \square 2) \square (q \square 0.5 \square q) \square n \square \left(\frac{\square}{k} \quad \frac{\square}{q} \right) \qquad (27)$$

The analysis of Equation (27) demonstrates that the principal factors for the reduction of the time required for the exponentiation is the number of bits concurrently processed during exponentiation - k and multiplication - q. It is also apparent that the dependence of T_E on q possesses an extremum, i.e. there exists an optimal value q_0 for

Original Article

which T_E is minimum. Given that q is integer, it is easy to determine the values of q_0 for the corresponding values of n used in practice. For n = 1024 the optimal value $q_0 = 5$ and for n = 2048 $q_0 = 6$. In this case the value of k is limited only by the available amount of memory for the storage of table T. Compared to the exponentiation in the Galois fields, with Montgomery reduction and without concurrent processing, the proposed procedure accelerates the computations by a factor of □ the arithmetic value of which is determined by the formula:

$$\frac{T_0}{T_E} = \frac{4.5 \cdot n^2}{2 \cdot q \cdot (q + 2^q + 2^{k-q} - 1.5) \cdot (2^k + 2) \cdot (q + 0.5 \cdot q) \cdot n \cdot (\xi + 2)} \quad (28)$$

It is apparent that for q = 1, implying that the concurrent processing of a group of bits is only used for squaring, the value of □ lies in the range 1.3 to 3. This means that the use of concurrent processing for the calculation of only a square in the Galois fields is ineffective. A much more significant acceleration in the calculation of the exponent in the Galois fields is obtained by the concurrent processing of the group of bits in both the squaring and the multiplication. In order to verify the theoretical results for the achieved acceleration of the calculation of the execution of exponentiation in the Galois fields i.e., for the calculation of $A|^E$ rem P, suitable simulations were performed. Table 7 shows the experimentally obtained values of ξ of the acceleration obtained during the calculation of the exponent in the Galois fields for different values of k. This table illustrates the achieved acceleration that approaches an order of magnitude. It additionally highlights the dependence of the calculation of the acceleration □ of the calculation of the exponent in the Galois fields on the group size k during squaring for the particular example of n = 2048 and q = 6.

Table 7. Experimentally achieved acceleration.

k□	□
6	4.78
7	5.33
8	5.81
9	6.26
10	6.67
11	7.05
12	7.41

The analysis shows that the efficiency of the proposed approach tends to decrease with an increase in the value of k due to the exponential growth of the volume V of the memory required for the table:

$$V \sim 2^k \cdot (n - k) + 2^q \cdot (n - q) \quad (29)$$

Hence for k = 10 and q = 6, the required memory space is V= 274 KBytes, a value that is feasible in most processing platforms, including microcontrollers.

VI. EVALUATION OF THE RESULTS

Modular exponentiation in Galois fields is a fundamental operation in various cryptographic applications, particularly in public key cryptography. This operation is essential for algorithms such as RSA, Diffie-Hellman, and ElGamal, where it facilitates secure key exchanges and digital signatures [17, 18, 19]. The efficiency of

modular exponentiation directly impacts the performance of these cryptographic systems, as it often involves repeated modular multiplications, which can be computationally intensive [19]. Techniques like Montgomery multiplication have been developed to optimize this process, enhancing both speed and security by minimizing vulnerabilities to side-channel attacks [17, 18]. Furthermore, the implementation of modular exponentiation in hardware architectures is crucial for achieving high performance in realworld applications, ensuring that cryptographic protocols can operate securely and efficiently in various environments [22, 23]. Overall, the role of modular exponentiation in Galois fields is pivotal for maintaining the integrity and confidentiality of cryptographic communications. The results of this ongoing research concern a range of mathematical operations. Firstly, a method for the fast calculation of the square in the Galois fields was theoretically founded, analyzed and developed. This method is based on the use of the polynomial property of the square, on the Montgomery group reduction in the Galois fields and on the use of preliminary calculations. It has been shown theoretically and experimentally that the use of the Montgomery group reduction with the processing of k bits concurrently, at the same time as using preliminary calculations, renders feasible the acceleration of the squaring in Galois fields by 0.75□k times. Following that, another method was developed for the accelerated multiplication by a constant number in the Galois fields. The acceleration of the calculation of the result is attained via the use of group reduction and preliminary calculations that combine the operations of addition of the multiplicand and the Montgomery correction. This rendered possible the acceleration of the multiplication by q times when q bits were concurrently processed. Based on the proposed methods for fast squaring and multiplication by a constant factor, a procedure was developed for the exponentiation in the Galois fields. The theoretical analysis and the experimental studies have demonstrated that their use is capable of significantly accelerating the computational application of this operation that is important for cryptographic applications. The acceleration is significant, of approximately one order of magnitude. The memory requirements for the implementation of the algorithms were calculated to be of the order of 10^2 kilobytes for numbers of 2048 bits, a value that is feasibly available even in the case of microcontrollers. By increasing the speed at which the calculation of the exponent takes place, the word length of the numbers for which the exponent can be effectively calculated is implicitly also increased. Hence, by enabling the use of numbers with larger numbers of digits, the cryptographic stability of the algorithms increased, together with the associated level of security.

VII. CONCLUSIONS AND FUTURE WORK

A collection of accelerated calculations was proposed that leads to the acceleration of the calculation of exponentiation in Galois fields. Galois Fields exponentiation is an operation that is fundamental for a wide spectrum of cryptographic algorithms. The ability to accelerate this calculation facilitates the use of strong cryptographic security in devices where it is otherwise difficult. This includes portable terminals, IoT and microcontroller-based systems. The approach consists of two developed procedures, namely fast exponentiation to the square and multiplication with a constant number in Galois fields. The acceleration was developed using the properties of the second order polynomial, the Montgomery group reduction and precalculations. The proposed method was founded mathematically. The operation of the proposed method was illustrated by simple arithmetic examples that were described in detail. The development was further supported by results obtained an implemented computer simulation that was used for deriving experimental results on the achieved acceleration. It has been proved, both theoretically and experimentally that the proposed approach renders possible the acceleration of exponentiation in Galois fields by 5 to 7 times, in comparison with known methods. The level of

Original Article

security provided by public-key cryptographic algorithms is completely determined by the word length of the numbers being processed. In practice, for many important applications, the number of bits is limited by the allowable time for the realization of the basic operation of public-key cryptography - exponentiation, the computational complexity of which has a cubic dependence on the length of numbers [13]. Accordingly, the achieved acceleration of exponentiation on Galois fields opens up opportunities for using numbers of larger digit capacity, i.e., increasing the level of resistance of publickey cryptographic algorithms. The proposed scheme will be an enabling technology for future research, aiming to exploit this proposal for achieving high levels of security in applications where this is limited. An initial target the implementation modular exponentiation with group processing of the exponent code and the use of precalculations that depend on the number that is raised to a power. This will facilitate the proliferation of the use of high-level security cryptographic primitives in smart-cards and microcontrollers with limited resources. A further application will involve the development of fast modular multiplication by a constant number, the length of which significantly exceeds the processor bit capacity, enabling increased levels of security in all types of processors. An additional target is the increasing the security level of the implementation of homomorphic encryption with modular exponentiation components. This will facilitate the accelerated implementation of this operation on IoT terminal devices via secure involvement of cloud computing resources. A further development will be the utilization of the proposed accelerated scheme for the definition of procedures for Fast Zero-Knowledge Identification Method, derived from of the well-known Schnorr schemes.

References

U. Jetzek, Galois Fields, Linear Feedback Shift Registers and Their Applications. Carl Hanser Verlag GmbH Co KG, 2018.

D. Canright, "A very compact S-box for AES," In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, pp. 441-455. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

J. Daemen, V. Rijmen, "The advanced encryption standard process," The Design of Rijndael: AES—the Advanced Encryption Standard (2002): 1-

8. Y. N. Shivani, A. Srinivas, B. K. Thanmayi, V. Vignesh, and B. V. Srividya, "EdDSA over Galois Field GF (p^m) for Multimedia Data," Journal of Engineering Research and Reports, vol. 4, no. 4, pp. 1-7, 2019.

J. Luo, K. D. Bowers, A. Oprea, and L. Xu, "Efficient software implementations of large finite fields GF (2 n) for secure storage applications," ACM Transactions on Storage (TOS), vol. 8, no. 1, pp. 1-27, 2012.

Nikolaos G. Bardis, O. P. Markovskyi, and N. Doukas, "A method for strict remote user identification using non-reversible Galois field transformations," In MATEC Web of Conferences, vol. 125, p. 05017, 2017.