TESTING OF PARALLEL CRYPTOGRAPHIC ALGORITHMS

TESTE DE ALGORITMOS CRIPTOGRÁFICOS PARALELOS

PRUEBA DE ALGORITMOS CRIPTOGRÁFICOS PARALELOS

Dzianis Bildziuk
bildziuk.d.m@mail.ru
http://orcid.org/0000-0002-7849-1589
Belarusian State University of Informatics and Radioelectronics, Belarus

Dmitry Mikhaylov
dmitry.m.mikhaylov@mail.ru
http://orcid.org/0000-0002-1593-8048
China Branch of BRICS Institute of Future Networks, China

Ilman Shazhaev
ilman.sh.shazhaev@mail.ru
http://orcid.org/0000-0002-6291-4591
Shanghai Jiao Tong University, China

Scientific Editor: José Edson Lara
Organization Scientific Committee
Double Blind Review by SEER/OJS
Received on 12/11/2021
Approved on 06/07/2022

ABSTRACT
The article considered the algorithms of the constituent primitive operations of asymmetric algorithms for cryptographic data transformation, which can be implemented in parallel computing systems (FPGA, GPGPU, quantum computers etc.) to accelerate cryptographic transformations. As a basis for the implementation of cryptographic primitives, it is proposed to use non-positional number systems. In particular, the article considered the residual number system and proposes two new - frequency and coordinate number systems, based on Fourier and Tom-Cook interpolation bijective mappings over a ring of integers, as well as the implementation of addition, multiplication and division with a remainder in these systems for the implementation of modular arithmetic at finite algebraic structures. The analysis of the computational and spatial complexity of cryptographic algorithms in suggested number systems is presented. As a result, the advantages of non-positional number systems are shown in the implementation of asymmetric algorithms for asymmetric cryptographic data transformation in parallel computing systems.

Keywords: public-key cryptography, parallel computing system, quantum computer, GPU, fast transform algorithms.

RESUMO
O artigo considerou os algoritmos das operações primitivas constituintes de algoritmos assimétricos para transformação de dados criptográficos, que podem ser implementados em sistemas de computação paralela (FPGA, GPGPU, computadores quânticos etc.) para acelerar transformações criptográficas. Como base para a implementação de primitivas criptográficas, propõe-se a utilização de sistemas numéricos não posicionais. Em particular, o artigo considerou o sistema de numeração residual e propõe dois novos - sistemas de numeração de frequência e de coordenadas, baseados em mapeamentos bijetivos de interpolação de Fourier e Tom-Cook sobre um anel de inteiros, bem como a implementação de adição, multiplicação e divisão com um restante nestes sistemas para a implementação de aritmética modular em estruturas algébricas finitas. A análise da complexidade computacional e espacial de algoritmos criptográficos em sistemas de numeração sugeridos é apresentada. Como resultado, as vantagens dos sistemas numéricos não posicionais são mostradas na implementação de algoritmos assimétricos para transformação de dados criptográficos assimétricos em sistemas computacionais paralelos.

Palavras-chave: criptografia de chave pública, sistema de computação paralela, computador quântico, GPU, algoritmos de transformação rápida.

RESUMEN
El artículo consideró los algoritmos de las operaciones primitivas constituyentes de los algoritmos asimétricos para la transformación de datos criptográficos, que pueden implementarse en sistemas de computación paralelos (FPGA, GPGPU, computadoras cuánticas, etc.) para acelerar las transformaciones criptográficas. Como base para la implementación de primitivas criptográficas, se propone utilizar sistemas numéricos no posicionales. En particular, el artículo consideró el sistema de números residuales y propone

dos nuevos sistemas de números de frecuencia y coordenadas, basados en mapeos biyectivos de interpolación de Fourier y Tom-Cook sobre un anillo de números enteros, así como la implementación de sumas, multiplicaciones y divisiones con un resto en estos sistemas para la implementación de aritmética modular en estructuras algebraicas finitas. Se presenta el análisis de la complejidad computacional y espacial de los algoritmos criptográficos en los sistemas numéricos sugeridos. Como resultado, las ventajas de los sistemas numéricos no posicionales se muestran en la implementación de algoritmos asimétricos para la transformación asimétrica de datos criptográficos en sistemas de cómputo paralelo.

Palabras clave: criptografía de clave pública, sistema de cómputo paralelo, computadora cuántica, GPU, algoritmos de transformación rápida.

1. INTRODUCTION

Public key cryptography algorithms are based on two mutually inverse one-way cryptographic functions. There is a class of cryptographic systems in which the implementation of these functions requires resource-consuming computational operations of exponentiation in a finite field or a residue class ring, e.g. the RSA algorithm. One can alternatively use algebraic-geometric cryptographic coding constructions, e.g. elliptic curve-based algorithms (ECC), Hermite curves or lattice on base of residue number system (RNS) with additive operation inside a group of points within algebraic-geometric area defined over a finite field.

However, when compared with symmetric key algorithms, the computational complexity of the abovementioned public key cryptography systems is relatively high and their implementation is resource consuming from the viewpoint of memory use and cryptographic processing time. One method of accelerating the execution of basic arithmetic operations for public key processing is paralleling. Paralleling includes designing parallel algorithms and their implementation in parallel computation systems.

2. METHODS AND THE DEVELOPMENT OF THE STUDY

One known classification of computer systems is Flynn's classification (Farber, 2011):

– *SISD* (single instruction stream / single data stream). This class includes serial computer systems having one central processor unit capable of processing only one stream of sequentially executed instructions.

– *MISD* (multiple instruction stream / single data stream). Theoretically, machines of this class should execute multiple instructions for a single data stream.

– *SIMD* (single instruction stream / multiple data stream). These systems usually include several processors which can execute one instruction for different data streams in lockstep mode.

– *MIMD* (multiple instruction stream / multiple data stream). These machines execute multiple instruction streams for multiple data streams.

In the class of *SIMD* computer systems, one can separate the *SMP* (symmetric multiprocessing) architecture systems the main distinctive feature of which is a common physical memory shared by all the system processors. These computation systems currently develop the most rapidly due to the use of graphics processor-based computation technologies (*GPGPU*, general-purpose graphics processing units, e.g. the *NVIDIA CUDA* technology). These systems are prototypes of quantum computation systems.

Quantum computation systems offer computational advantage provided a *SIMD* type parallel computation algorithm has been developed, and therefore the development and testing of these algorithms is currently an important task (Thiel, 2021).

Furthermore, quantum computation systems (and the availability of fast parallel computation algorithms) are a threat for the cryptographic security of existing algorithms (e.g. *AES* or the El Gamal encryption system in an *ECC* additive group). Then another important task is to develop algorithms to be secure against quantum computations (Srivastava, 2021): since a quantum computation system is a lattice in some sense (from the viewpoint of mapping-of-sets cryptographic function implementation), the implementation of these cryptographic algorithms should be based on the lattice theory. One promising approach is the solution of the shortest vector and closest vector problems (*SVP* and *CVP*) on the basis of the lattice theory (Divesh) with non-positional number system base (Bajard & Imbert, 2004).

Parallel algorithms are algorithms part of which can be executed simultaneously by different processor units. This definition implies the utilization of any of the above described parallel computation systems.

A parallel algorithm can be described with a pseudo code containing a notion of the number of parallel processes to be executed for each operation. Also known as the connected

graph description method where the graph nodes are defined as separate operations. If the result of one operation is used by another operation, then their respective nodes are connected with a graph edge. The simultaneously executed operations are represented in the form of nodes at the same level; thus, the height of a parallel algorithm graph defined as the number of its levels characterizes the graph execution time.

If n operations out of the N possible ones are executed in sequence (i.e., if n is the height of the graph), then the parameter $\beta = n/N$ is the share of sequential operations. Obviously, $\beta = 1$ for *SISD* computation systems. The maximum achievable acceleration R for paralleling between l processor units as compared with sequential graph execution is described by Amdahl's law:

$$R = 1/(1 - P + P/l), \quad (1)$$

where $P = 1 - \beta$ is the part of algorithm execution time that can be paralleled between l processor units.

The number of processor units l used by parallel algorithms (i.e. the width of the respective graph) characterizes the resource consumption of graph execution, e.g. power consumption and memory use. Then another important parameter of a parallel algorithm is its efficiency determined as the ratio of acceleration and the number of parallel processor units used: $S = R/l$.

However, a computational problem can be solved using various algorithms involving different numbers of computation operations N (different computational complexity), and these algorithms may have different shares of sequential operations β. From the viewpoint of computation speed, the better of two algorithms is the one with the lowest graph height n or with the shortest execution time τ for implementation on a specific computer. Efficiency-based comparison between such algorithms is not indicative because the algorithms have different R. Therefore the absolute efficiency indicator can be accepted to be the parameter equal to the area of the computational graph (the product of the height n and the width l of the graph), or the cost of the parallel algorithm, i.e., the more efficient algorithm is the one that has the higher speed and uses the smaller number of parallel processor units. Thus, the cost of

the parallel graph is $C = nl$ or $C = \pi l$ for algorithm execution utilizing a specific computational platform.

To compare the efficiency of asymmetric algorithms for cryptographic data transformation in parallel computing systems, implementations based on the CUDA technology were used. El Gamal based on elliptic curves and Shorr cryptosystems were considered as test asymmetric cryptographic systems - the most popular algorithms today used in most cryptosystems (for example, EMV banking systems (EMVCo, 2011), cryptocurrency systems (Hartwig, 2016)). Comparative analysis of the implementation efficiency was carried out on the basis of a theoretical assessment of cost and execution time.

2.1 Basic Prime Field Operations in Cryptographic Systems

The basic operations of cryptographic systems that implement operations directly over a prime field of integers are defined as the operations of summation, multiplication, exponentiation and multiplicative modular inversion of big prime integers (modular operations). An example is the Schnorr digital signature algorithm (a variant of the El Gamal cryptographic system family) illustrated in Fig. 1 (Katz, 2014):

Input: Element $g \in GF(p)^*$, $ord(g) = q$. Private key $x \in GF(p)^*$ (Public key defined as $y \in GF(p)^*$, $y \leftarrow g^x \bmod p$). Message $m \in GF(*)$. Random integer $l \in GF(q)$. Two big prime integers p и q, $q \mid p-1$. Cryptographic function $H : GF(*) \rightarrow GF(q)$.

Output: Digital signature s.

1: $r \leftarrow g^l \bmod p$;

2: $e \leftarrow H(m \| r)$;

3: $s \leftarrow (l - xe) \bmod q$;

Return s.

Figure 1. Schnorr digital signature algorithm

The elliptic curve E over the field $GF(q)$ is the smooth curve set by the equation of the following type:

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6, a_i \in GF(q), \quad (2)$$

referred to as the Weierstrass form. We introduce the notation $E(X,Y)$ for a set of points $(x, y) \in (GF(q) \times GF(q))$ satisfying Eq. (2) and further containing an infinitely remote point

denoted as O. The finite field $GF(q)$ can generally be extended $GF(p^{\square})$ where p is a prime integer.

The scalar multiplication of a point of an elliptic curve in *ECC* based systems is based on the two-point summation group operation (Katz, 2014). The pseudo-code of the general point summation algorithm $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ is shown in Fig. 2.

Input: Elliptic curve coefficients (see Eq. 1), points $P_1 = (x_1, y_1)$ (or $P_1 = O$) and $P_2 = (x_2, y_2)$ (or $P_2 = O$).

Output: Point $P = P_1 + P_2$.

1: **if** $P_1 = O$ **then** $P \leftarrow P_2$ **elif** $P_2 = O$ **then** $P \leftarrow P_1$ **elif** $P_2 = -P_1$ **then** $P \leftarrow O$ **else** $P = -(x_3, y_3)$;

Return P.

Where $P = -(x_3, y_3) = (x_3, \tilde{y}_3)$, $\tilde{y}_3 = -a_1 x_3 - a_3 - y_3$:

- if $x_1 \neq x_2$, then accept $\lambda = (y_2 - y_1)/(x_2 - x_1)$, $\beta = y_1 - \lambda x_1$ and reduce $E(X, \lambda X + \beta) = 0$ to the form $X^3 + \gamma X^2 + aX + b = 0$, $\gamma, a, b \in GF(q)$ and calculate $x_3 = -\gamma - x_1 - x_2$, $y_3 = y_1 + \lambda(x_3 - x_1)$;
- else ($P_1 = P_2 = (x, y)$) use formal differentiation rules to obtain $E'(X,Y) = -\frac{dE(X,Y)/dX}{dE(X,Y)/dY}$, calculate $\lambda' = E'(x, y)$, $\beta = y - \lambda' x$ and reduce $E(X, \lambda' X + \beta) = 0$ to the form $X^3 + \gamma' X^2 + aX + b = 0$, $\gamma', a, b \in GF(q)$ and calculate $x_3 = -\gamma' - 2x$, $y_3 = y + \lambda'(x_3 - x)$.

Figure 2. General point summation algorithm $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ for elliptic curve $E(X,Y)$ in finite field $GF(q)$

It can be seen from the point summation algorithm shown above that the basic operations of *ECC* based cryptographic systems are modular operations of summation, multiplication and multiplicative inversion in finite field $GF(q)$.

The basic operation of the *NTRU* algorithms (Albrecht, 2016) for the lattice L is the cyclic convolution in polynomial ring $Z_q[X]/(X^N - 1)$ requiring implementation of modular operations, by analogy with the group operations with Hermite curves (Shi, 2015) in finite field $GF(q)$.

Then one way to reduce the cryptographic operation execution time for asymmetric cryptographic systems is to develop fast parallel algorithms of four operations: summation, multiplication, calculation of the multiplicative inverse over finite field and integer division

with a remainder. All these operations can be implemented either in positional or in non-positional number systems.

2.2 Big integer representation in different number systems

Secure public key cryptography systems over $GF(q)$ utilize large-dimension fields ($p > 2^{160}$ for *ECC*). Then the modular operations utilized in cryptographic processing are implemented as operations with big integers based on the number of registers in the processor units used.

Positional number system

One known method of big integer representation is the B-based representation in the positional number system (PNS - (Jha, 2020)). The integer $a > B$ in the PNS is the integer vector $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)_B$, such that $a = \sum_{i=0}^{n-1} a_i B^i$, $0 \le a_i < B$. Alternatively, the integer a in the PNS can be interpreted as the polynomial $a(x) = \sum_{i=0}^{n-1} a_i x^i$, where a_i are the vector representation coefficients. Typically, the base B is chosen to be equal to a power of two, due to the binary logic of the existing processor units. Then the operations of division, multiplication and calculation of the remainder of division by a power of two are executed by shifting and truncation of the high order part of the integer, respectively.

The magnitude of the base B in the PNS determines the length of the vector representing the respective integer; this affects the execution speed of arithmetic operations (for parallel computation systems, it affects the number of processor units used) as well as the memory size used. An increase in B leads to a reduction in the length of the vector $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)_B$, but the hardware implementation of arithmetic operations in processor units is limited to the machine word register length. Then $B = 2^m$ is limited by the number of processor unit registers and, furthermore, the magnitude of the base should accommodate for possible overflow of the processor unit registers resulting from the execution of interim arithmetic operations.

The maximum carry may occur as a result of the multiplication of two integers $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)_B$ and $\mathbf{b} = (b_{n-1}, b_{n-2}, \ldots, b_0)_B$ –

$\max a_i \times \max b_j = (B-1)\times(B-1) = (B^2 - 2B + 1) = B(B-2)+1 = (B-2,1)_B$. Thus, for efficient integer representation in the PNS the bit representation of the base *B* should not be greater than half of the number of processor unit registers: for example, for 64-bit processor unit the magnitude of the base should not exceed 32 binary bits ($B \le 2^{32}$).

Mixed base positional number system

A generalization of the PNS is the system with the base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_0)$ referred to as the mixed base positional number system (MPNS) (Bi, 2008). Any integer $0 \le a < M$ in this system is the vector $\mathbf{a} = (a_{r-1}, a_{r-2}, \ldots, a_0)_M$, $0 \le a_i < m_i$, such that $a = a_0 + \sum_{i=1}^{r-1}\left(a_i \prod_{j=0}^{i-1} m_j\right)$. The MPNS has the same mathematical operations as the PNS. However, the shift operations specify division and multiplication along with the base shift. For example, the operation of division by $d = \prod_{i=0}^{l-1} m_i$ is set by the *l*-tuple right shift with the new integer representation base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_l)$, and if the number of digits $a_i \ne 0$ for $i = 0,1,\ldots,l-1$, then the division result will be less accurate. The multiplication of the integer *a* by $d = \prod_{i=0}^{l-1} m_i$ with the base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_l)$ is executed through *l*-tuple left shifting with the new integer representation base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_0)$, where the number of digits $a_i = 0$ for $i = 0,1,\ldots,l-1$. The remainder of integer *a* division by $d = \prod_{i=0}^{l-1} m_i$ is determined by truncation (resetting) of the high order part of the integer $(a_{r-1}, a_{r-2}, \ldots, a_l)$.

Coordination number system

The search for fast multiplication (convolution) algorithms of big integers has led to the emergence of non-positional number systems on the basis of interpolation functions. For example, the Karatsuba-Ofman algorithm reduces the number of digit-by-digit multiplications from n^2 to $n^{\log_2 3}$ by interpreting two big integers $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)_B$ and $\mathbf{b} = (b_{n-1}, b_{n-2}, \ldots, b_0)_B$ as the integers $a = a_1' B^{n/2} + a_0'$ and $b = b_1' B^{n/2} + b_0'$, and their product as

$$c = ab = a'_1 b'_1 B^n + ((a'_1 + a'_0)(b'_1 + b'_0) - a'_1 b'_1 - a'_0 b'_0) B^{n/2} + a'_0 b'_0, \quad (3)$$

or

$$c = ab = a'_1 b'_1 (B^n + B^{n/2}) + (a'_1 - a'_0)(b'_1 - b'_0) B^{n/2} + a'_0 b'_0 (B^{n/2} + 1). \quad (4)$$

The Toom-Cook algorithm allows partitioning the factors $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)_B$ and $\mathbf{b} = (b_{n-1}, b_{n-2}, \ldots, b_0)_B$ into r + 1 parts (the Karatsuba-Ofman algorithm is a particular case of the Toom-Cook algorithm for r = 1 (Knuth, 1997)), i.e. $a = \sum_{i=0}^{r} a_i B^{\frac{n}{r+1} i}$ and $b = \sum_{i=0}^{r} b_i B^{\frac{n}{r+1} i}$. Then $c = ab = c'(B^{n/(r+1)})$ with further alignment of the vector $\mathbf{c} = (c_{2n-1}, c_{2n-2}, \ldots, c_0)_B$, where $c'(x) = a'(x)b'(x) = \sum_{i=0}^{2r} c'_i x^i$, $a'(x) = \sum_{i=0}^{r} a'_i x^i$, $b'(x) = \sum_{i=0}^{r} b'_i x^i$. The coefficients of the polynomial $c'(x)$ are restored on the base of 2r + 1 interpolation points such that $c'(k) = a'(k)b'(k)$, $k = 0,1 \ldots 2r$:

$$c'_i = w_{i,0} - c'_{i+1} \cdot \Delta^i, \quad (5)$$

where $\Delta^i = i(i+1)/2$ is the sum of the first i natural integer, $c'_{2r} = w_{2r,0}$, $w_{i,j} = \left(w_{i-1,j+1} - w_{i-1,j}\right) / i$, $w_{0,j} = c'(j)$, $i = 0,1,\ldots,2r$, and $j = 0,1,\ldots,2r - i$.

The complexity of the Toom-Cook algorithm is estimated as $O\left(n^{1+\varepsilon}\right)$ digit-by-digit multiplications.

It should be noted that this algorithm is also suitable for the summation and subtraction of big integers, and the coefficients of the polynomial $c'(x) = a'(x) \pm b'(x) = \sum_{i=0}^{r} c'_i x^i$ can be restored on the base of r + 1 interpolation points such that $c'(k) = a'(k) \pm b'(k)$, $k = 0,1 \ldots r$. Then one can define a coordination number system (CNS) where any integer a can be represented in the form of two points of the Cartesian plane on the basis of the PNS polynomial representation.

The integer a in the CNS with the base $\mathbf{K} = (k_n, k_{n-1}, \ldots, k_0)$ is represented by the vector $\mathbf{a}' = (a'_n, a'_{n-1}, \ldots, a'_0)_{K,B}$ relative to the PNS with the base B such that $a'_i = a(k_i) = \sum_{j=0}^{n-1} a_j k_i^j$, where $a(x) = \sum_{j=0}^{n-1} a_j x^j$ is the polynomial representation of a in the PNS $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)_B$. The inverse

conversion from the CNS to the PNS can be implemented with well-known interpolation methods (based on the Lagrange interpolation polynomial, Newton's iterations etc.).

One disadvantage of the CNS is the overflow of the processor unit registers at the interpolation points. For example, the integer $2^{2047} < a < 2^{2048}$ (e.g. in the *RSA* cryptographic system) is represented in the PNS with $B = 2^{32}$ in the form of a 64-digit integer or a 63th order polynomial, while in the CNS this integer should be represented by 64 interpolation points. The value of the polynomial $a(x)$ at the 64th point will be represented by an integer greater than 2^{384} which will require more than 12 x 32-bit registers and lead to the formation of a new additional CNS for that point. The Toom-Cook algorithm resolves such situations by recursion or branching parallel processes.

One more disadvantage of this algorithm is the absence of integer comparison and division operations. Equity checking is also not always implementable because the same integer can be represented with different polynomialы in the PNS, e.g. an integer represented by convolution (multiplication) of two other integers $c(x) = a(x)b(x)$ and the same integer with carry operations executed after convolution $c'(x)$ ($c'(x) \neq c(x)$, but $c'(B) = c(B)$). For the same reason division with remainder of the integer $c(x)$ by the integer $d(x)$ is only possible if the dividend is represented by the points of the polynomial $c(x) = q(x)d(x) + r(x)$, i.e., for the CNS with $r = c \bmod d$ and $c(k_i) = a(k_i)b(k_i)$ the polynomial $r(x)$ cannot generally be restored from the points $r(k_i) = c(k_i) \bmod d(k_i)$, $k_i \in \mathbf{K}$.

The multidimensional coordination number system (MCNS) can be defined by analogy with the CNS relative to the MPNS. For example the integer a in the MPNS with the base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_0)$ can be represented in the form of the multidimensional polynomial $a(x_0, x_1, \ldots, x_{r-2}) = a_0 + \sum_{i=1}^{r-1}\left(a_i \prod_{j=0}^{i-1} x_j \right)$, such that $a = a(m_0, m_1, \ldots, m_{r-2})$. Then this integer in the MCNS is determined by the values of the multidimensional plane at the interpolation points, that is, the vector $\mathbf{a}' = (a'_n, a'_{n-1}, \ldots, a'_0)_{K,M}$, where $a'_i = a(\mathbf{k}_i)$ is the value at the point $\mathbf{k}_i = (k_{i,0}, k_{i,1}, \ldots, k_{i,r-2})$. The base of this MCNS is described by the array of vectors $\mathbf{K} = (\mathbf{k}_n, \mathbf{k}_{n-1}, \ldots, \mathbf{k}_0)$ or the matrix $\mathbf{K} = [k_{i,j}]$, $i = 0,1,\ldots,n$, $j = 0,1,\ldots,r-2$. The number of

interpolation points *n* is determined by the multidimensional function interpolation algorithm (e.g. the Ben-or/Tiwary algorithm, the Prony method etc. (Kaltofen, 2000)).

The MCNS possesses all the properties (including disadvantages) of the C NS except the rapid growth of its values at the interpolation points. Selecting the CNSS base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_0)$ such that all the values m_i are different and close to 2^{32}, one can expect the integer $2^{2047} < a < 2^{2048}$ to be represented in the form of a 64-digit integer in the MPNS. Then limiting the CNS base $\mathbf{K} = [k_{i,j}]$ to a binary field (in order to reduce the values at the multidimensional polynomial points) one can limit the maximum number of interpolation points to 2^{63}.

Frequency interpolation number system

Fast convolution computation algorithms further include discrete Fourier transformation (DFT) ones. For this algorithm, the interpolation points are powers of the primitive root □ (the DFT core), and the $a(\omega^i)$ values at these points (for the polynomial representation of the integer *a* in the PNS with the base *B*) are the frequency representation (spectrum) of the integer *a* calculated by direct DFT. Then the inverse DFT is the interpolation function. A widely used fast convolution computation algorithm is the Schönhage-Strassen algorithm based on the DFT in the integer ring *Z / mZ*. However, faster is the Fuhrer algorithm which is the improved Schönhage-Strassen algorithm with multiple compress of big integer (Fürer, 2007).

Then the frequency interpolation number system (FNS) over the set *Z* of integers is determined based on the parameters of the DFT used for the FNS to PNS conversion and vice versa. For example the integer *a* represented in the *B*-ary PNS in the form of the vector $\mathbf{a}' = (a'_{n-1}, a'_{n-2}, \ldots, a'_0)_B$ or the polynomial $a'(x) = \sum_{j=0}^{n-1} a'_j x^j$ will be represented in the FNS by the vector $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)_{\omega,m}$ (or the polynomial $a(x) = \sum_{j=0}^{n-1} a_j x^j$), such that $\mathbf{a} = F(\mathbf{a}')$, where *F* is the *n*-point Fourier transformation over a set of integers and □ is the primitive *n*-th root of unity by module *m* (*Z / mZ* ring element) (Knuth, 1997). Each *i*-th coordinate of the

vector **a** is defined by the transformation $a_i = a'(\omega^i) \bmod m = \sum_{j=0}^{n-1} a'_j \omega^{ij} \bmod m$, and for the vector $\mathbf{a}'$, by the transformation $a'_i = n^{-1} a(\omega^{-i}) \bmod m = n^{-1} \sum_{j=0}^{n-1} a_j \omega^{-ij} \bmod m$, and then $a = \sum_{i=0}^{n-1} \left(n^{-1} a(\omega^{-i}) \bmod m \right) B^i$, $0 \le a < B^n - 1$.

The FCNS defines the summation, subtraction and multiplication operations $c(\omega^i) = a(\omega^i) b(\omega^i)$, $c(\omega^i) = a(\omega^i) \pm b(\omega^i)$ and inherits all the FNS disadvantages. It should be further noted that the integer vectors in the FCNS cannot have an arbitrary length n determined by the parameters of the DFT used ($\omega^n \bmod m = 1$). Also, the module m in the FCNS is limited to the half of the processor unit registers (for integer multiplication) and, pursuant to the primitive root theorem, if 128-bit (binary) integers are used, it cannot be less than 64 bits. Thus, an increase in the bit number in the PNS leads to a rapid growth of the modulus and requires additional FCNS for some orders of digits in integers. The Schönhage-Strassen algorithm is known to calculate a cyclic convolution using two primitive roots $\omega^n \bmod m = 1$ and $\psi^{2n} \bmod m = 1$, $\omega = \psi^2$, thus allowing one to reduce the dimension of the Z / mZ ring. For a 64-bit processor unit the maximum PNS base is determined as $B = 2^{16}$, and then the module is $m = 2^{32} + 1$ (product of digit orders in the FNS does not exceed the number of processor unit registers).

Fast DFT algorithms (FFT) providing for rapid PNS to FNS conversion and vice versa include algorithms based on the DFT computation through multidimensional DFT (e.g. the Cooley–Tukey and the Good-Thomas algorithms (Shirbhate, 2015)).

The Fuhrer algorithm employs the representation of the source factors in the form of multidimensional polynomials thus defining the representation in the multidimensional frequency number system (MFNS). The maximum dimension of the Z / mZ rings (in the multidimensional FFT) and the total number of the interpolation points remain the same as in the ordinary FNS, and then the only advantage is the computational complexity of the MFNS to PNS and vice versa conversions.

Residue class number systems

Fast convolution computation algorithms further include the Chinese remainder theorem (CRT) based ones, i.e. the Winograd algorithm for polynomials, the A. Schönhage algorithm for integers and the Agarwal-Cooley algorithm for convolution computation (Valueva, 2020).

The CRT based fast integer multiplication algorithm has a relationship with the algorithms the CNS is based upon. For example, the calculation of the value of the polynomial $a(x)$ at the point □ for FCNS can be treated as $a(\omega^i) = a(x) \bmod (x - \omega^i)$, while in CRT-based integer algorithms any integer $a < M$ can be restored from the points $a_i = a \bmod m_i$, with $M = m_0 m_1 \ldots m_{r-1}$, m_i being mutually prime integers.

Thus, the residue class number system (RNS) is determined by the product of mutually prime integers $M = m_0 m_1 \ldots m_{r-1}$ (RNS base), and then the integer $0 \le a < M$ is the residue vector $\mathbf{a} = (a_0, a_1, \ldots, a_{r-1})_M$ such that $a = \sum_{i=0}^{r-1} a_i M_i N_i \bmod M$, $M_i = M / m_i$, $N_i = M_i^{-1} \bmod m_i$, $a_i = a \bmod m_i$.

The main advantage of the RNS is its independence on the polynomial integer representation in the PNS. The summation/subtraction or multiplication result for two integers $\mathbf{a} = (a_0, a_1, \ldots, a_{r-1})_M$ and $\mathbf{b} = (b_0, b_1, \ldots, b_{r-1})_M$ in the RNS is the vector $\mathbf{c} = (c_0, c_1, \ldots, c_{r-1})_M$, $0 \le c < M$, where $c_i = a_i \pm b_i \bmod m_i$ or $c_i = a_i b_i \bmod m_i$, respectively. The integer division in the RNS is only implementable for a zero remainder, i.e. $b \mid a$, $c_i = a_i b_i^{-1} \bmod m_i$. Comparison is only possible in the RNS for equity checking between two integers, $a_i \equiv b_i \bmod m_i$, $i = 0,1,\ldots r-1$.

To eliminate the disadvantages of the RNS for integer division with non-zero remainder, MPNS with the same base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_0)$ is used. There are efficient (including parallel) algorithms for integer conversion from RNS to MPNS as well as integer scaling algorithms for RNS by means of this conversion (Lyubomudrov, 2014). By analogy with the PNS, the maximum length of the binary representation of the integers m_i in the RNS and MPNS should not exceed half of the number of processor unit registers for storing interim results in a single register. However, from the viewpoint of RNS integer representation vector

length reduction (aimed at reducing the number of parallel processor units required) one should choose as big m_i as possible (e.g. $2^{31} < m_i < 2^{32}$ for a 64-bit processor unit).

2.3 Search criteria of fast cryptographic processing in a finite field

We will hereinafter assume that the integers *a* and *b* in any number system are vectors of the same length and the operations with these integers are executed in the finite prime field *GF*(*p*). Then *a* and *b* are random equiprobable integers less than *p*.

Cryptographic operations can be implemented either in one number system or with conversion to other number systems. Then the search for the most efficient cryptographic function implementation algorithm can be formalized as the search for the lowest weight path in the graph shown in Fig. 3.

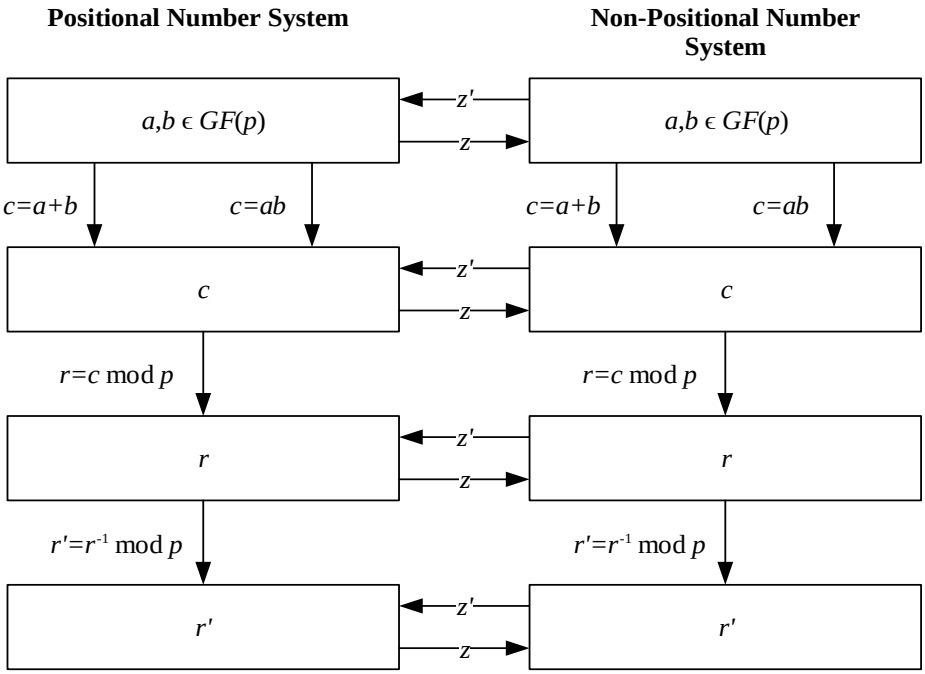

Figure 3. Graph of possible implementations of cryptographic function operations in the field *GF*(*p*).

The weight of each edge of the graph in Fig 1 (*z* and *z'* are conversions between number systems) can be estimated for paralleling purposes in accordance with the selected criteria (speed, cost etc.). Hence the search of the most efficient path requires weighing each

edge of the graph. An edge that is absent in the cryptographic function should have a zero weight.

We will compare two parallel algorithms for four number systems: PNS with the base $B = 2^{32}$, MPNS with the base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_0)$, where $m_i < 2^{32}$ are the biggest prime integers, RNS with the base equal to that of the MPNS and FNS with the base $B = 2^{16}$ of the related PNS and the $m = 2^{32} + 1$ dimension of the *Z/mZ* ring.

2.4 Fast parallel big integer summation algorithm

The summation of two big integers $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)_B$ and $\mathbf{b} = (b_{n-1}, b_{n-2}, \ldots, b_0)_B$ in the PNS with the base B is a digit-by-digit summation of the elements of their vector representation $a_i + b_i, i = 0,1,\ldots n-1$ followed by alignment of the result. The maximum carry value for the summation is unity: $\max a_i + \max b_i = (B-1) + (B-1) = B + B - 2 = (1, B-2)_B$. Then the length of the result vector of the integer $c = a + b$ should be greater by 1 than the length of the vectors **a** and **b**, i.e. the result vector is $\mathbf{c} = (c_n, c_{n-1}, \ldots, c_0)_B$. If the summation terms belong to the finite field *GF*(*p*), then based on the same maximum carry assumptions the condition $B^{n+1} > 2p - 2$ should be satisfied.

The parallel alignment process (overflow carry to higher orders) for an integer vector in the PNS is described by the function *AlignPNS*(**c**):

$\mathbf{c} \leftarrow AlignPNS(\mathbf{c}): \mathbf{c} \rightarrow ($

$q_i \leftarrow \lfloor c_i / B \rfloor$; $\quad for \forall i \in \{0,1,\ldots n-1\}$

if $q_i > 0$ **then** $f \leftarrow 1$ **else** $f \leftarrow 0$ **end if**; $\quad for \forall i \in \{0,1,\ldots n-1\}$

while $f = 1$ **do**

$c_i \leftarrow c_i \bmod B$; $\quad for \forall i \in \{0,1,\ldots n-1\}$

$c_{i+1} \leftarrow c_{i+1} + q_i$; $\quad for \forall i \in \{0,1,\ldots n-1\}$

$q_i \leftarrow \lfloor c_i / B \rfloor$ $\quad for \forall i \in \{0,1,\ldots n-1\}$

if $q_i > 0$ **then** $f \leftarrow 1$ **else** $f \leftarrow 0$ **end if**; $\quad for \forall i \in \{0,1,\ldots n-1\}$

end do;

return c;

).

The maximum number of carry cycles in Eq. **Erro! Fonte de referência não encontrada.** is *n* (it is always *n* for sequential execution) if the summation result is equal or greater than *B*.

The parallel summation algorithm in the PNS can be expressed in the form of the function *AddPNS*(**a**,**b**):

$$\begin{aligned}
&\mathbf{c} \leftarrow AddPNS(\mathbf{a},\mathbf{b}) : (\mathbf{a},\mathbf{b}) \rightarrow (\\
&\quad c_i \leftarrow a_i + b_i; && for\ \forall i \in \{0,1,\ldots n-1\}\\
&\quad \mathbf{return}\ AlignPNS(\mathbf{c});\\
&).
\end{aligned} \tag{6}$$

The terms $\mathbf{a} = (a_{r-1}, a_{r-2}, \ldots, a_0)_M$ and $\mathbf{b} = (b_{r-1}, b_{r-2}, \ldots, b_0)_M$ in the MPNS with the base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_0)$ have approximately the same dimension as in the PNS (8192-bit integers $m_i < 2^{32}$ (biggest prime integers) require 257 and 256 registers, respectively). However, integer alignment takes into account the integer-order index and is described by the function *AlignMPNS*(**c**):

$$\begin{aligned}
&\mathbf{c} \leftarrow AlignMPNS(\mathbf{c}) : \mathbf{c} \rightarrow (\\
&\quad q_i \leftarrow \lfloor c_i / m_i \rfloor; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\quad \mathbf{if}\ q_i > 0\ \mathbf{then}\ f \leftarrow 1\ \mathbf{else}\ f \leftarrow 0\ \mathbf{end\ if}; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\quad \mathbf{while}\ f = 1\ \mathbf{do}\\
&\qquad c_i \leftarrow c_i \bmod m_i; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\qquad c_{i+1} \leftarrow c_{i+1} + q_i; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\qquad q_i \leftarrow \lfloor c_i / m_i \rfloor; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\qquad \mathbf{if}\ q_i > 0\ \mathbf{then}\ f \leftarrow 1\ \mathbf{else}\ f \leftarrow 0\ \mathbf{end\ if}; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\quad \mathbf{end\ do};\\
&\quad \mathbf{return}\ \mathbf{c};\\
&).
\end{aligned} \tag{7}$$

The parallel summation algorithm in the MPNS can be expressed in the form of the function *AddMPNS*(**a**,**b**):

$$\begin{aligned}
&\mathbf{c} \leftarrow AddMPNS(\mathbf{a},\mathbf{b}) : (\mathbf{a},\mathbf{b}) \rightarrow (\\
&\quad c_i \leftarrow a_i + b_i; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\quad \mathbf{return}\ AlignMPNS(\mathbf{c});).
\end{aligned} \tag{8}$$

Parallel summation of two integers $\mathbf{a} = (a_0, a_1, \ldots, a_{r-1})_M$ and $\mathbf{b} = (b_0, b_1, \ldots, b_{r-1})_M$ in the RNS with the result vector $\mathbf{c} = (c_0, c_1, \ldots, c_{r-1})_M$ is set by the function *AddRNS*(**a**,**b**):

$$\begin{aligned}
&\mathbf{c} \leftarrow AddRNS(\mathbf{a},\mathbf{b}) : (\mathbf{a},\mathbf{b}) \rightarrow (\\
&\quad c_i \leftarrow (a_i + b_i) \bmod m_i; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\quad \mathbf{return}\ \mathbf{c};).
\end{aligned} \tag{9}$$

The product of the base elements $M = m_0 m_1 \ldots m_{r-1}$ for the summation of two integers of the $GF(p)$ field should satisfy the inequality $M > 2p - 2$.

In the FNS the summation is executed by analogy with the RNS, i.e., the result is the vector $\mathbf{c} = (c_{n-1}, c_{n-2}, \ldots, c_0)_{\omega,m}$, where $c_i = a_i + b_i \bmod m$. Summation in the FNS is set by the function $AddFNS(\mathbf{a},\mathbf{b})$:

$$\begin{aligned} &\mathbf{c} \leftarrow AddFNS(\mathbf{a},\mathbf{b}) : (\mathbf{a},\mathbf{b}) \rightarrow (\\ &\quad c_i \leftarrow (a_i + b_i) \bmod m; \qquad\qquad for\ \forall i \in \{0,1,\ldots n-1\} \\ &\quad \mathbf{return c};). \end{aligned} \tag{10}$$

We will now pre-estimate the speed and cost (the height and area of the parallel graph) for the summation operations in different number systems for the worst scenario (maximum number of operations in the height of the parallel graph). To this end we will make the calculations for the terms of the $GF(p)$ field represented in the PNS with the base B. Two random integers $a, b \in GF(p)$ in the PNS are two n-digit integers with the result being an $n + 1$ digit integer. Then the longest (critical) path for the parallel function as per Eq. **Erro! Fonte de referência não encontrada.** includes $5n + 2$ elementary computer operations, and for the function as per Eq. **Erro! Fonte de referência não encontrada.** it includes $5n + 3$ elementary operations. The maximum number of parallel streams in the PNS is $n + 1$ taking into account that the result occupies one additional digit, and hence the cost of the function as per Eq. **Erro! Fonte de referência não encontrada.** is $(5n+3)(n+1) = 5n^2 + 8n + 3$ operations $\times$ streams. Two similar integers $a, b \in GF(p)$ in the MPNS will in the worst scenario be $n + 1$ digit integers, and the result will be an $n + 2$ digit integer. Then the critical path for the function as per Eq. **Erro! Fonte de referência não encontrada.** is $(5n+3)(n+2) = 5n^2 + 18n + 16$ operations $\times$ streams. If the RNS has the same base as the MPNS then the number of digits in the terms remains the same ($n + 1$) but the base should in this case be chosen such that the result be not greater than the product of the base elements, i.e., the source integers and the result will consist of $n + 2$ digits. Then the critical path for the function as per Eq. **Erro! Fonte de referência não encontrada.** consists of 2 operations (summation and modular operation) and its cost is $2(n + 2) = 2n + 4$ operations $\times$ streams. The PNS related with the FNS has the base $B' = B / 2$ (if $B' = 2^{16}$ then $m = 2^{32} + 1$), and then the

number of digits in the integers in the FNS (taking into account that the result is one digit longer) is $2n + 1$. Thus, the critical path for the function as per Eq. **Erro! Fonte de referência não encontrada.** is 2 operations and the cost is $2(2n+1) = 4n + 2$ operations × streams. Summation speed and cost estimates are presented in Table 1.

Table 1.

Number of critical path operations and cost of summation for different number systems for *n*-digit terms $a, b \in GF(p)$ in the PNS with the base *B*.

	PNS	MPNS	RNS	FNS
Critical path, operations		$5n + 8$	2	2
Cost, operations × streams	$5n^2 + 8n + 3$	$5n^2 + 18n + 16$	$2n + 4$	$4n + 2$

Source: research data

2.5 Fast parallel big integer multiplication algorithm

The result of multiplication of two big integers $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)_B$ and $\mathbf{b} = (b_{n-1}, b_{n-2}, \ldots, b_0)_B$ in the PNS is the double-length vector $\mathbf{c} = (c_{2n-1}, c_{2n-2}, \ldots, c_0)_B$ (based on the maximum carry for two-digit multiplication).

A widely used PNS multiplication algorithm is "column" multiplication which is a linear convolution of two vectors $c_i = \sum_{j=0}^{n-1} a_j b_{i-j}$ and $0 \le i \le 2n-1, c_{2n-1} = 0$ followed by result alignment. Parallel implementation of this algorithm requires execution of each of the n^2 parallel multiplications $a_{i \bmod n} b_{\lfloor i/n \rfloor}$ in a separate stream and result added to the digit position $c_{i \bmod n + \lfloor i/n \rfloor}$ followed by carry in the vector $\mathbf{c} = (c_{2n-1}, c_{2n-2}, \ldots, c_0)_B$ in accordance with the base *B*. This algorithm is expressed by the function *MulPNS*(**a**,**b**):

$$\begin{aligned} &\mathbf{c} \leftarrow MulPNS(\mathbf{a},\mathbf{b}) : (\mathbf{a},\mathbf{b}) \rightarrow (\\ &\quad c_{i \bmod n + \lfloor i/n \rfloor} \leftarrow c_{i \bmod n + \lfloor i/n \rfloor} + a_{i \bmod n} \cdot b_{\lfloor i/n \rfloor}; \qquad for\ \forall i \in \{0,1,\ldots n^2 - 1\} \\ &\quad \textbf{return}\ AlignPNS(\mathbf{c});). \end{aligned} \tag{11}$$

It should be specially noted that adding the result of each elementary multiplication to the respective digit position of the total result leads to a queue of additions. The length of the queue of *n* additions is the maximum for the digit position c_{n-1}, which adds *n* operations to the

critical path of the function as per Eq. **Erro! Fonte de referência não encontrada.**. Furthermore, one should bear in mind the possibility of processor unit register overflow in case of a long queue of additions and hence reduce the allowed dimension of the base *B*. For example, a 64-bit processor unit may use the base $B = 2^{32}$ for storing elementary multiplication results, and taking into account the addition of *n* such multiplication results, the maximum allowed base dimension is $B = 2^{32-\left\lceil \frac{\log_2 n}{2} \right\rceil}$.

In the MPNS with the base $\mathbf{M} = (m_{2r-1}, m_{2r-2}, \ldots, m_0)$ the multiplication of two integers $\mathbf{a} = (a_{r-1}, a_{r-2}, \ldots, a_0)_M$ and $\mathbf{b} = (b_{r-1}, b_{r-2}, \ldots, b_0)_M$ can also be described as a convolution followed by result alignment but taking into account the coefficient $\mathbf{K} = (k_{r^2-1}, k_{r-2}, \ldots, k_0)$ for the shift of one of the factors for "column" multiplication (because $m_i \neq m_j$ for $i \neq j$), where $k_i = \frac{\prod_{x=0}^{i \bmod r} m_x \cdot \prod_{y=0}^{\lfloor i/r \rfloor} m_y}{\prod_{z=0}^{i \bmod r + \lfloor i/r \rfloor} m_z}$. Then parallel multiplication in the MPNS is described by the function *MulMPNS*(**a**,**b**):

$$\begin{aligned} &\mathbf{c} \leftarrow MulMPNS(\mathbf{a},\mathbf{b}) : (\mathbf{a},\mathbf{b}) \rightarrow (\\ &\quad c_{i \bmod r + \lfloor i/r \rfloor} \leftarrow c_{i \bmod r + \lfloor i/r \rfloor} + a_{i \bmod r} \cdot b_{\lfloor i/r \rfloor} \cdot k_i; \qquad for\ \forall i \in \{0,1,\ldots r^2 - 1\} \\ &\quad \textbf{return}\ AlignMPNS(\mathbf{c});). \end{aligned} \tag{12}$$

To ensure accurate results the coefficients k_i should be stored in the form of simple fractions (for direct multiplication by a coefficient, accuracy is ensured by carrying to higher digit orders) while the numerator and the denominator may be integers exceeding the number of processor unit registers (e.g. if the base elements are prime integers). Obviously, Eq. **Erro! Fonte de referência não encontrada.** has an excessive computational complexity and will be further excluded from the comparison.

In the RNS, multiplication is expressed by the function *MulRNS*(**a**,**b**):

$$\begin{aligned}&\mathbf{c} \leftarrow MulRNS(\mathbf{a},\mathbf{b}):(\mathbf{a},\mathbf{b}) \rightarrow (\\ &\quad c_i \leftarrow (a_i b_i) \bmod m_i; \qquad for\ \forall i \in \{0,1,\ldots r-1\}\\ &\quad \mathbf{return} \mathbf{c};).\end{aligned} \tag{13}$$

In the FNS, the multiplication algorithm is designed by analogy with that for the RNS and is set by the function *MulFNS*(**a**,**b**):

$$\begin{aligned}&\mathbf{c} \leftarrow MulFNS(\mathbf{a},\mathbf{b}):(\mathbf{a},\mathbf{b}) \rightarrow (\\ &\quad c_i \leftarrow (a_i b_i) \bmod m; \qquad for\ \forall i \in \{0,1,\ldots n-1\}\\ &\quad \mathbf{return} \mathbf{c};).\end{aligned} \tag{14}$$

By analogy with the summation operations, we will theoretically estimate the speed and cost of multiplication in different number systems for the worst scenario and the critical path of the functions Eq. **Erro! Fonte de referência não encontrada.**, **Erro! Fonte de referência não encontrada.** and **Erro! Fonte de referência não encontrada.**, given the representation of *n*-digit factors in the PNS with the base $B = 2^{32}$ implemented on a 64-bit processor unit. Since the function as per Eq. **Erro! Fonte de referência não encontrada.** may cause overflow of the processor unit registers for c_{n-1} (due to addition of elementary multiplication results) the factors should be represented in the PNS with the base $B' = 2^{32-\log_2 n}$, i.e., the required number of digits for the factors is $n' = \lceil 32n/(32-\log_2 n) \rceil$, and the result is a 2*n*'–digit integer. Then the maximum critical parallel multiplication path in the PNS (taking into account carry operations) is 11*n*' + 3 elementary operations and the cost is $(11n'+3)n'^2 = 11n'^3 + 3n'^2$ operations × streams. For the RNS the factors are *n* + 1 digit integers and the result is a 2(*n* + 1) digit one. Since the RNS base should be preset the factors and the result will be determined by the number of digits in the result. Hence the critical path for the function as per Eq. **Erro! Fonte de referência não encontrada.** is 2 and the cost is $2 \cdot 2(n+1) = 4n+4$ operations × streams. For the FNS the base is half of *B* hence the factors are 2*n*-digit integers and the result is a 4*n*-digit one. Then the critical path for the function as per Eq. **Erro! Fonte de referência não encontrada.** is 2 operations (same as for the RNS) and the cost is $2 \cdot 4n = 8n$ operations × streams. Multiplication speed and cost estimates are presented in Table 2.

Table 2

Number of critical path operations and cost of multiplication for different number systems for *n*-digit factors $a,b \in GF(p)$ in the PNS with the base *B*.

	PNS	I NS	NS
Critical path, operations	$11\left\lceil \frac{32n}{32-\log_2(n)} \right\rceil + 3$	2	2
Cost, operations × streams	$11\left\lceil \frac{32n}{32-\log_2(n)} \right\rceil^3 + 3\left\lceil \frac{32n}{32-\log_2(n)} \right\rceil^2$	$4n+4$	$8n$

Source: research data

Fast parallel algorithm for big integer division with remainder

Integer a division by p, $a > p$, is set by an expression like $a = qp + c$ where a is the dividend, p is the divider, q is the quotient and c is the remainder, $c < p$. It is also stated that a is comparable with c by modulus of p, i.e. $a \equiv c(\mathrm{mod}\, p)$ or $c = a \bmod p$.

Summation and multiplication in the finite field $GF(p)$ are defined by the modulus of p. Let the summation or multiplication result of two elements of $GF(p)$ be a, then the final result in the preset finite field is $c = a \bmod p$.

If a is the result of the summation of two integers of $GF(p)$, i.e. $0 \le a < 2p$, then the residue is determined by the function $Rem(a, p)$;

$$c \leftarrow Rem(a, p) : (a, p) \rightarrow \begin{cases} a - p, & p \le a < 2p \\ a, & 0 \le a < p \end{cases} \tag{15}$$

In any number system Eq. **Erro! Fonte de referência não encontrada.** is implemented via one comparison and, if necessary, one subtraction. In the PNS and MPNS integers are compared digit-by-digit, beginning from the highest order down to the first differing digits of two big integers. Comparison can be combined with subtraction for implementing Eq. **Erro! Fonte de referência não encontrada.**. Then the difference $c = a - p$ is first calculated; if $c > 0$ then this is the result, otherwise the result is the initial integer a. Parallel implementation of Eq. **Erro! Fonte de referência não encontrada.** in PNS is set by the function $RemPNS(\mathbf{a},\mathbf{p})$:

$$
\begin{aligned}
&\mathbf{c} \leftarrow RemPNS(\mathbf{a},\mathbf{p}) : (\mathbf{a},\mathbf{p}) \rightarrow (\\
&\quad c_i \leftarrow a_i - p_i; && for\ \forall i \in \{0,1,\ldots n-1\}\\
&\quad f \leftarrow 1;\\
&\quad \textbf{while}\ f = 1\ \textbf{do}\\
&\qquad \textbf{if}\ c_i < 0\ \textbf{then}\\
&\qquad\quad c_i \leftarrow c_i + B; && for\ \forall i \in \{0,1,\ldots n-1\}\\
&\qquad\quad c_{i+1} \leftarrow c_{i+1} - 1; && for\ \forall i \in \{0,1,\ldots n-1\}\\
&\qquad \textbf{end if};\\
&\qquad \textbf{if}\ c_i < 0\ \textbf{then}\ f \leftarrow 1\ \textbf{else}\ f \leftarrow 0\ \textbf{end if}; && for\ \forall i \in \{0,1,\ldots n-1\}\\
&\quad \textbf{end do};\\
&\quad \textbf{if}\ c_n < 0\ \textbf{then}\ \mathbf{c} \leftarrow \mathbf{a}\ \textbf{end if};\\
&\quad \textbf{return}\ \mathbf{c};)
\end{aligned}
\tag{16}
$$

Parallel implementation of Eq. **Erro! Fonte de referência não encontrada.** is set by the function *RemMPNS*(**a**,**p**):

$$
\begin{aligned}
&\mathbf{c} \leftarrow RemMPNS(\mathbf{a},\mathbf{p}) : (\mathbf{a},\mathbf{p}) \rightarrow (\\
&\quad c_i \leftarrow a_i - p_i; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\quad f \leftarrow 1;\\
&\quad \textbf{while}\ f = 1\ \textbf{do}\\
&\qquad \textbf{if}\ c_i < 0\ \textbf{then}\\
&\qquad\quad c_i \leftarrow c_i + m_i; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\qquad\quad c_{i+1} \leftarrow c_{i+1} - 1; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\qquad \textbf{end if};\\
&\qquad \textbf{if}\ c_i < 0\ \textbf{then}\ f \leftarrow 1\ \textbf{else}\ f \leftarrow 0\ \textbf{end if}; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\quad \textbf{end do};\\
&\quad \textbf{if}\ c_r < 0\ \textbf{then}\ \mathbf{c} \leftarrow \mathbf{a}\ \textbf{end if};\\
&\quad \textbf{return}\ \mathbf{c};)
\end{aligned}
\tag{17}
$$

Direct implementation of Eq. **Erro! Fonte de referência não encontrada.** in the FNS and RNS is impossible due to the absence of the comparison operation.

For FNS one should use inverse DFT for comparison in PNS, but this conversion does not yield a result with negative interpolation polynomial coefficients. Then subtraction in the FNS is not always possible (including the case $a > p$): this means that the implementation of Eq. **Erro! Fonte de referência não encontrada.** in the FNS should be completely executed in the PNS (i.e. using Eq. **Erro! Fonte de referência não encontrada.**) with the utilization of

inverse and direct DFT. Based on the abovementioned disadvantages (dependence of FNS representation on PNS representation), implementation of Eq. **Erro! Fonte de referência não encontrada.** in the FNS will not be considered hereinafter.

The RNS has a system of orthogonal bases (based on the Chinese remainder theorem) allowing scaling an integer with an expanded set of bases; this system allows comparing integers in the RNS via the MPNS. Integer representation in the RNS is invariant to representation in other number systems, and this property allows implementing Eq. **Erro! Fonte de referência não encontrada.** using orthogonal bases and Eq. **Erro! Fonte de referência não encontrada.** without the inverse conversion to the RNS. For example, if the RNS with the base $M = m_0 m_1 \ldots m_{r-1}$ is defined, then the system of orthogonal bases is the matrix of coefficients $\mathbf{W} = [w_{ij}]$, $i, j = (0,1,\ldots,r-1)$ where each line $\mathbf{w}_i$ is interpreted as the vector integer representation MN_i / m_i, ($N_i = (M / m_i)^{-1} \bmod m_i$) in the MPNS with the base $\mathbf{M} = (m_{r-1}, m_{r-2}, \ldots, m_0)$. Then the parallel function algorithm for Eq. **Erro! Fonte de referência não encontrada.** in the RNS is defined by the function *RemRNS*(**a,p,p'**) where **a** and **p** are the vector representations of the dividend and the divider in the RNS and **p'** is the vector representation of the divider in the MPNS:

$$
\begin{aligned}
&\mathbf{c} \leftarrow \mathit{RemRNS}(\mathbf{a},\mathbf{p},\mathbf{p}') : (\mathbf{a},\mathbf{p},\mathbf{p}') \rightarrow (\\
&\quad f \leftarrow 0;\\
&\quad \textbf{if } a_i \neq p_i \textbf{ then } f \leftarrow 1 \textbf{ end if}; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\quad \textbf{if } f = 0 \textbf{ then}\\
&\qquad c_i \leftarrow 0; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\quad \textbf{else}\\
&\qquad a'_{i \bmod r} \leftarrow a'_{i \bmod r} + a_{\lfloor i/r \rfloor} b_{\lfloor i/r \rfloor, i \bmod r}; && for\ \forall i \in \{0,1,\ldots r^2-1\}\\
&\qquad \textbf{if } \mathit{RemMPNS}(\mathbf{a}',\mathbf{p}') = \mathbf{a}' \textbf{ then}\\
&\qquad\quad c_i \leftarrow a_i; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\qquad \textbf{else}\\
&\qquad\quad c_i \leftarrow a_i - p_i \bmod m_i; && for\ \forall i \in \{0,1,\ldots r-1\}\\
&\qquad \textbf{end if};\\
&\quad \textbf{end if};\\
&\quad \textbf{return } \mathbf{c};)
\end{aligned}
\qquad (18)
$$

Theoretical estimation of Eq. **Erro! Fonte de referência não encontrada.** for the integer *a* which is a result of summation of two *n*-digit integers of *GF*(*p*) (in the PNS with the base $B = 2^{32}$) in the number systems considered (see Eqs. **Erro! Fonte de referência não encontrada.**, **Erro! Fonte de referência não encontrada.** and **Erro! Fonte de referência não encontrada.**) for the worst critical path scenario is presented in Table 3.

Table 3
Number of critical path operations and cost of computing residue for summation in different number systems for *n*-digit terms $a, b \in GF(p)$ in the PNS with the base *B*.

	PNS	MPNS	RNS
Critical path, operations	$4n+7$	$4n+11$	$4n+16$
Cost, operations × streams	$4n^2+11n+7$	$4n^2+19n+22$	$4n^3+32n^2+80n+64$

Source: research data

If the integer *a* is the result of multiplication of two integers of *GF*(*p*) one should use integer division with remainder. Three main methods of integer division with remainder are known based on Fermat's descent, Newton's iterations and Montgomery transformation (Bajard & Imbert, 2004).

In the PNS the Fermat descent-based division is implemented in the form of the widespread "column" division algorithm (Knuth, 1997). It has been reported that in the general case Newton iteration and Montgomery transformation based PNS algorithms provide no advantage in computational complexity but if the divider *p* is known *a priori* the computational complexity can be reduced by carrying out preliminary calculations. Fermat descent-based integer division is set by the function *DivFerma* (*a, p*):

$$
\begin{aligned}
&c \leftarrow DivFerma(a,p):(a,p)\rightarrow(\\
&\quad c \leftarrow a;\\
&\quad \textbf{while}\, c > p\, \textbf{do}\\
&\qquad c \leftarrow c - qp, qp \approx c, c \geq 0;\\
&\quad \textbf{end do};\\
&\quad \textbf{return c};)
\end{aligned}
\tag{19}
$$

The main task in the implementation of Eq. **Erro! Fonte de referência não encontrada.** is the search for such components of the quotient *q* to provide for iteration

approach to the remainder *c* through the minimal number of iterations. The number of iterations depends on the factor by which the dividend is greater than the divider, and if that factor is large, only one iteration may be sufficient, while if the quotient is unity Eq. **Erro! Fonte de referência não encontrada.** degenerates to Eq. **Erro! Fonte de referência não encontrada.**. Newton iteration and Montgomery transformation based division is invariant to the ratio of the dividend and the divider. For example, the calculation of the remainder of the division $c = a \bmod p$ with the value of $p' = (-p)^{-1} \bmod M$ pre-calculated on the basis of Montgomery transformation is determined by the function *DivMont* (*a*, *p*, *p'*):

$$c \leftarrow DivMont(a, p, p') : (a, p, p') \rightarrow Rem\left(\frac{a + (ap' \bmod M)p}{M}, p \right) \tag{20}$$

The value $M > p$ is chosen so division by it to have lower computational complexity than the division by *p* (e.g. using shift operations). Newton iteration based division requires calculation of the interpolated quotient and then the remainder: for $M > p^2$ and the pre-calculated value of $p' = \lfloor M / p \rfloor$ the integer division is described by the function *DivNewton* (*a*, *p*, *p'*):

$$c \leftarrow DivNewton(a, p, p') : (a, p, p') \rightarrow Rem\left(a - \left\lfloor \frac{ap'}{M} \right\rfloor p, p \right) \tag{21}$$

In the PNS Eq. **Erro! Fonte de referência não encontrada.** is implemented in the form of D. Knuth's algorithm and set by the parallel function $\mathbf{c} \leftarrow DivFerma(\mathbf{a}, \mathbf{p})$.

$$
\begin{aligned}
&\mathbf{c} \leftarrow DivFermaPNS(\mathbf{a},\mathbf{p}) : (\mathbf{a},\mathbf{p}) \rightarrow (\\
&\quad d \leftarrow 1; \\
&\quad \textbf{if } p_{n-1} < \lfloor B/2 \rfloor \textbf{ then } d \leftarrow \lceil B/2p_{n-1} \rceil \textbf{ end if}; \\
&\quad p'_j \leftarrow p_j d; && for\ \forall\ j \in \{0,1,\ldots n-1\} \\
&\quad AlignPNS(\mathbf{p}'); \\
&\quad m \leftarrow 0; \\
&\quad \textbf{if } a_j \neq 0 \textbf{ and } j > i \textbf{ then } m \leftarrow j \textbf{ end if}; && for\ \forall\ j \in \{0,1,\ldots 2n-1\} \\
&\quad m \leftarrow m+1; \\
&\quad c_j \leftarrow a_j; && for\ \forall\ j \in \{0,1,\ldots m+n-1\} \\
&\quad c_{m+n} \leftarrow 0; \\
&\quad \textbf{for } i \textbf{ from } m+n \textbf{ to } n \textbf{ do} \\
&\qquad c'_{i-j} \leftarrow c'_{i-j}\, d; && for\ \forall\ j \in \{0,1,\ldots n\} \\
&\qquad AlignPNS(\mathbf{c}'); \\
&\qquad \textbf{if } c_i = p_{n-1} \textbf{ then} \\
&\qquad\quad q \leftarrow B-1; \\
&\qquad \textbf{else} \\
&\qquad\quad q \leftarrow \lfloor c_i B + c_{i-1} / p_{n-1} \rfloor; \\
&\qquad \textbf{end if}; \\
&\quad \textbf{end do};
\end{aligned}
\tag{22}
$$

3. COMPARATIVE ANALYSIS

The asymmetric cryptographic algorithms shown in Figure 1 and Figure 2 were implemented using the CUDA technology in the PNS, RNS and FNS number systems with parallel basic operations (as considered above). The results of measuring the performance and cost of algorithms on the NVIDIA GeForce RTX 2070 video card are presented in Figure 4 and Figure 5.

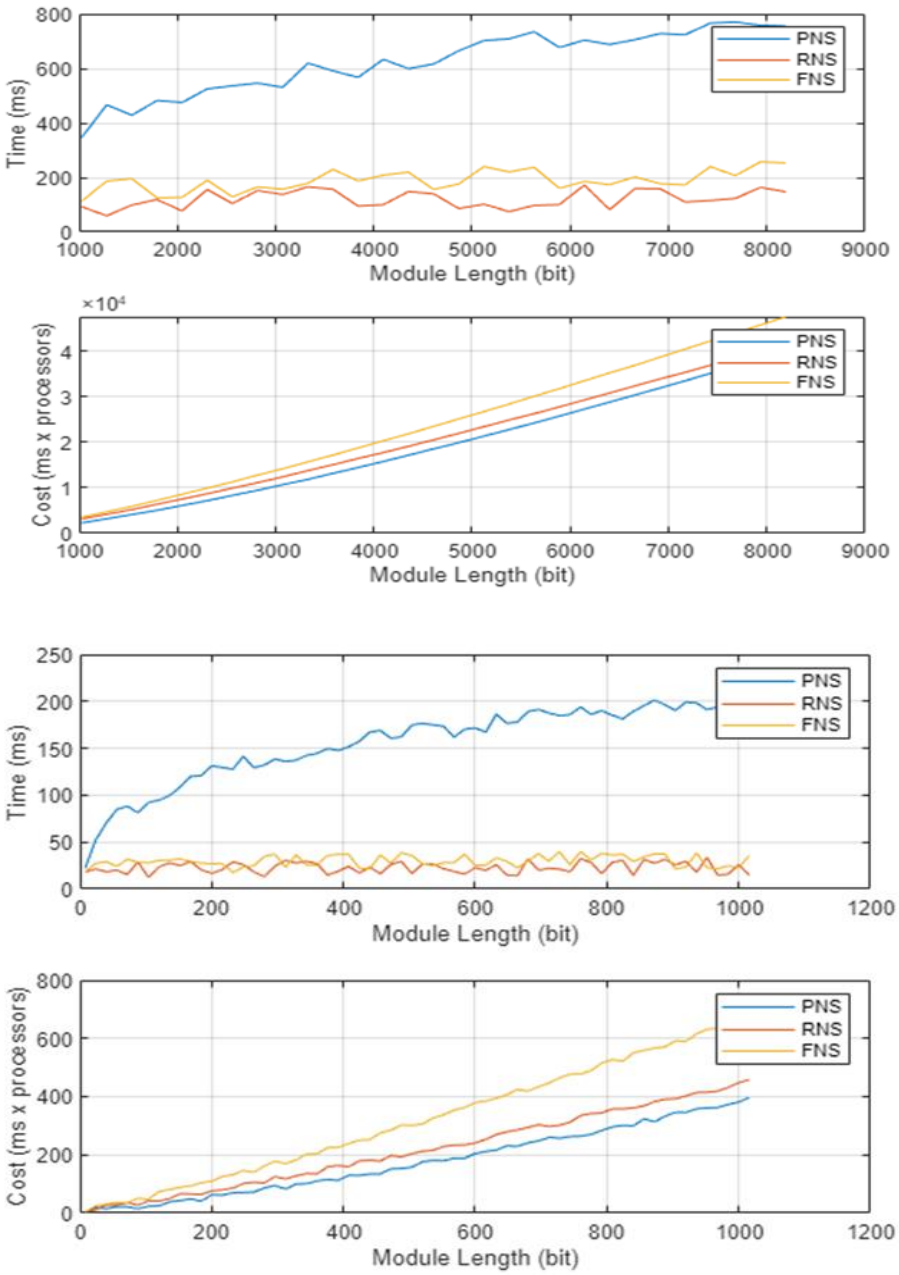

Figure 5. General point summation algorithm $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ for elliptic curve $E(X,Y)$ in finite field $GF(q)$

Source: research data

As can be seen from the presented measurements, the cost of the algorithms is about the same, however, in terms of speed, interpolation number systems are obviously in the lead. The most optimal is the use of RNS as the fastest numbering system with low cost.

4. CONCLUSION

Thus, the presented research is of practical interest for the implementation of asymmetric cryptographic algorithms based on interpolation number systems in parallel computing systems.

The presented parallel basic arithmetic operations make it possible to implement asymmetric cryptographic algorithms within one non-positional number system without returning to the positional one, in contrast to the algorithms presented, for example, in (Bajard & Imbert, 2004) or (Salamat, 2021).

The generalization of RNS on the example of FNS allows us to speak about the existence of interpolation number systems, which, in turn, makes the search for new, more efficient, non-positional number systems promising.

REFERENCES

Albrecht, M. B. (2016). A subfield lattice attack on overstretched NTRU assumptions. CRYPTO, 9814, 153-178. doi:https://doi.org/10.1007/978-3-662-53018-4_6

Bajard , J., & Imbert, I. (2004). A full RNS implementation of RSA. IEEE Transactions on Computers, 769-774.

Bi, S. &. (2008). The mixed-radix Chinese remainder theorem and its applications to residue comparison. IEEE Transactions on Computers, 1624-1632.

Divesh, A. (n.d.). Dimension-Preserving Reductions Between SVP and CVP in Different p-Norms. Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA). doi:https://doi.org/10.1137/1.9781611976465.145

EMVCo. (2011, November 29). Book 2: Security and Key Management. Version 4.3. Integrated Circuit Card Specifications for Payment Systems.

Farber, R. (2011). CUDA Application Design and Development. Burlington: Elsevier Science.

Fürer, M. (2007). Faster Integer Multiplication. Proceedings of the 39th annual ACM Symposium on Theory of Computing (STOC), 55-67.

Hartwig, M. (2016). ECDSA Security in Bitcoin and Ethereum: a Research Survey. CoinFabrik, 50.

Jha, A. C. (2020). Positional Number System. NUTA Journal, 1-9.

Kaltofen, E. L. (2000). Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm. Proceedings of the 2000 international symposium on Symbolic and algebraic computation, 192-201.

Katz, J. (2014). Introduction to modern cryptography: principles and protocols (2 ed.). Chapman and Hall.

Knuth, D. (1997). The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Third Edition. Addison-Wesley, 65–66, 208–209, and 290.

Lyubomudrov, A. A. (2014). The method for converting numbers represented in a positional number system into the residue number system. Vestnik Natsional'nogo Issledovatel'skogo Yadernogo Universiteta MIFI, 252-253.

Salamat, S. (2021). Fast and Energy Efficient Big Data Processing on FPGAs. UC San Diego Electronic Theses and Dissertations.

Shi, X. (2015). A novel memristive electronic synapse-based Hermite chaotic neural network with application in cryptography. Neurocomputing 166, 487-495.

Shirbhate, R. P. (2015). Design of parallel FFT architecture using Cooley Tukey algorithm. International Conference on Communications and Signal Processing (ICCSP) , 574-578.

Srivastava, V. (2021). Cryptanalysis of LRainbow: The Lifted Rainbow Signature Scheme. Provable and Practical Security. doi:https://doi.org/10.1007/978-3-030-90402-9_16

Thiel, C. a. (2021). Quantum Computer Resistant Cryptographic Methods and Their Suitability for Long-Term Preservation of Evidential Value. BLED. Retrieved from https://aisel.aisnet.org/bled2021/30

Valueva, M. (2020). Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. Mathematics and Computers in Simulation, 177, 232-243. doi:https://doi.org/10.1016/j.matcom.2020.04.031.