МЕТОДЫ, МЕХАНИЗМЫ И АЛГОРИТМЫ КРИПТОГРАФИЧЕСКИХ ПЕРСПЕКТИВНЫХ ПРЕОБРАЗОВАНИЙ

UDC 004.056.55

I.D. GORBENKO, Doctor of technical sciences, O.G. KACHKO, Ph.D. of technical sciences, M.V. YESINA, Ph.D. of technical sciences

GENERAL STATEMENTS AND ANALYSIS OF THE END-TO-END ENCRYPTION ALGORITHM NTRU PRIME IIT UKRAINE

Introduction

Today the questions concerning the stability of existing cryptographic algorithms to quantum cryptanalysis become topical. This is due, first of all, to the rapid development in the field of quantum computers. Therefore, it is necessary to evaluate the possibilities of quantum cryptanalysis and, on this basis, to modify existing cryptographic algorithms (for example, to increase the size of key parameters) or to create new cryptographic algorithms that will be resistant to attacks on quantum computers.

So, in light of the foregoing, NIST USA has announced a competition for the post-quantum algorithms search, including end-to-end encryption algorithms (E2EE) [7]. It is definitely known that for practical application algorithms must meet the requirements of stability, performance and should be low-resource. Submissions were received by NIST until November 30, 2017. They relate to: E2EE asymmetric algorithms and electronic signature (ES). Subsequently, their detailed analysis and comparison is expected, with a period of up to 3 years. This indicates the significant complexity of the problem to be solved.

With the participation of the authors of this article for the NIST USA competition, a cryptographic algorithm for NTRU Prime IIT Ukraine [8, 9], developed using NTRU [1] and NTRU Prime [2], was presented.

This article describes the differences between the proposed cryptographic algorithm and ANSI standard [1] and the NTRUPrime algorithm [2]. In each paragraph, attention is paid to the difference, the possibility of optimization, and the results of the research.

All experiments to determine the computational complexity were performed on the Intel(R) Core(TM) processor i5-4440 CPU @ 3.10 GHz. The spent time is determined in the processor tacts.

The objective of this paper is a general overview and description of the proposed cryptographic transformation end-to-end encryption «NTRU Prime IIT Ukraine», implementation specificity, comparison of the main characteristics and indicators, as well as the definition of differences from existing NTRU-like cryptographic algorithms.

1. Denotations and abbreviations

Most of the terms and denotations are the same as those adopted in the ANSI [1]. For convenience, we give them in this document.

Table 1

Denotations and abbreviations, that are used

db	The number of bits for the random component that is used during encryption. It is determined by cryptostability. Coincides with the length of the used hash.
n	Polynomial order. It determines the number of its coefficients. Prime for which the polynomial $x^n–x–1$ is irreducible.
q	The module, prime, by which the polynomial coefficients are given in $(Z/qz)[X](x^n–x–1)$; $q \geq 48t+3$.
F	Polynomial in $(Z/3z)[X](x^n–x–1)$. Specifies the private key.
G	Polynomial in $(Z/3z)[X](x^n–x–1)$. It is used to calculate the public key.
R	Blinding polynomial in $(Z/3z)[X](x^n–x–1)$.
f	Polynomial, which is calculated using the formula: $f=3F+1$.
h	Polynomial in $(Z/qz)[X](x^n–x–1)$. Calculated by the formula $h=3G/f$.
t	It determines the number of non-zero elements in the private key and message. For F, the number of 1 and -1 is $2t$, for G the number of 1 is $n/3+1$, and the number of -1 is $n/3$, for r – the number of 1 and -1 is $n/3$, for message the number of 0, 1 and -1 is not less than t.

Continuation of Table 1

Q	The module, prime, by which the polynomial coefficients are given in $(Z/qz)[X](x^n-x-1)$; $q\geq48t+3$.
E_p	Encrypted message, the polynomial in $(Z/qz)[X](x^n-x-1)$.
E_b	Encrypted message, bytes string.
Hlen	The length of the hash (bit) coincides with *db*.
k	Security Level.
m	Message to encrypt, bytes string.
M	Message after addition the random string and other information. It is used to encryption and decryption.
MGF	Mask generation function
qBits	Number of bits to specify the number *q*.
maxMsgLenBytes	Maximum length of the message.

2. Basic and additional parameters

2.1 Basic parameters

k – Security level. It is defines the remaining parameters. In [2], this parameter is indicated by λ.

n –polynomial order. It determines the number of its coefficients. The prime for which the polynomial x^n-x-1 is irreducible. In [2], this parameter is indicated by *p*.

t – determines the number of non-zero elements in a private key and message. For *F*, the number of 1 and –1 is 2*t*, for *G* the number of 1 and –1 are $2n/3+1$, and the number of –1 is $n/3$, for *r* – the number of 1 and –1 is $n/3$, for message the number of 0, 1 and –1 is not less than *t*.

q – the module in calculating the coefficients of the polynomial in $(Z/qz)[X](x^n-x-1)$. A prime number that satisfies the condition $q\geq48t+3$.

Further, as parameters (*n, q, t*) are selected parameters from [2].

Exceptions:

1. According to the requirement of no decoding errors, parameters *q*, *t* must satisfy the requirement $q\geq48t+3$. According to [2] these parameters satisfied the requirement $q\geq48t+3$. Due to the change in requirement, some values of the *q* parameter have been changed. The parameters that have been changed are shown in the Table 2. The first column defines the parameter numbers in the Table B Parameters [2]. Numbering from 0.

Table 2

Values of changed parameters

№	It was			It has become		
	n	*Q*	*T*	*n*	*q*	*t*
5	463	6481	135	463	6529	135
29	587	5233	109	587	5237	109
87	823	4513	94	823	4519	94
97	881	3217	67	881	3221	67

2. For some parameters from [2], the values of 3*t* are approximately equal to *n*. The use of such parameters during encryption leads to the need for repeated execution of the encryption operation with various random data that is supplemented the message. This is due to the fact that the probability of obtaining a polynomial in which the number of 1, –1, 0 does not satisfy the requirement to remain at least *t* is quite large. Repeated execution of the encryption operation greatly increases the time of its execution.

In Table 3 are the values of parameters that are *not recommended* for use on this basis. The first column defines the parameter numbers in the Table B Parameters [2]. Numbering from 0.

Table 3

Values of parameters that are not recommended for use

№	N	Q	t	$3t$	$n–3t$
2	461	7607	153	459	2
3	461	8779	153	459	2
7	463	9371	154	462	1
11	491	8627	163	489	2
12	491	9277	163	489	2
13	499	8243	166	498	2
14	499	9029	166	498	2
16	503	8663	167	501	2
24	557	9323	185	555	2
32	599	9551	198	594	4

2.2 Additional parameters

qBits – number of bits for q, $qBits = \lceil \log_2 q \rceil$. It is used by functions of transforming a polynomial in $(Z/qz)[X](x^n–x–1)$ to the byte string, if the same number of bytes is used for each element. If you use a method of minimizing the key length, the parameter is not used.

db – the number of bits for the random component. It is calculated by the formula:

$$db = \begin{cases} 128 & k < 128 \\ 192 & k < 192 \\ 256 & k < 256 \end{cases}$$

For k>200 db=256.

maxMsgLenBytes – determines the message maximum length. It is determined by the polynomial with small coefficients length. In encoding 2 polynomial coefficients are replaced by the bit string of 3 bits long. Thus, the total length of the bit string is (n–1)/2*3/8 bytes. This string should contain a random component in the length of db/8 bytes, the message length is 1 byte and the message itself. That is, the message maximum length is calculated by the formula

$$maxMsgLenBytes=(n–1)/2*3/8–db/8–1;$$

bc, *bk* are constants for calculating the module by the Barret method;

c – the number of bits used to determine the polynomial non-zero element index (see IGF2 [1]). It is determined by n:

$$c = \begin{cases} 9 & n < 512 \\ 10 & n < 1024 \\ 11 & n \geq 1024 \end{cases}$$

dm0 – determines the number of non-zero elements in the encoded polynomial $dm0=t$;

Hlen – hash length

$$Hlen = \begin{cases} 160 & k \leq 112 \\ 256 & k > 112 \end{cases}$$

minCallsMask – determines the number of the hash function calls for the algorithm (MGF_TP-1). It is determined by the formula:

$$minCallsMask=(16*n+HashLenBits*5–1)/(HashLenBits*5)+1;$$

minCallsR – determines the number of the hash function calls for the algorithm (IGF-2). It is determined by the formula: $minCallsR=(t*4*c+HashLenBits)/HashLenBits$;

pkLen – the number of public key bits that are used to form the string for encryption; *pkLen=db*

OID – 3 *байта*, *OID*[0]=0; *OID*[1]=1; *OID*[2]=2;

3. Key generation

3.1 Key data

Private key consists of:
Polynomial G in $(Z/3z)[X](x^n–x–1)$, the number of non-zero elements is equal to $2n/3+1$;
Polynomial F in $(Z/3z)[X](x^n–x–1)$, the number of non-zero elements is equal to $2t/3$;
Polynomial f=3F+1 in $(Z/qz)[X](x^n–x–1)$;
Polynomial f^{-1} in $(Z/qz)[X](x^n–x–1)$;
Public key – polynomial h in $(Z/qz)[X](x^n–x–1)$.

3.2 Polynomial with a given number of nonzero elements generation algorithm

Small polynomial Generation (*SmallPolynomialGeneration*)
Component The parameters *n*
Input Not zero items count *count*, random numbers generator *rand*
Output the polynomial *dest* of degree *n*–1
The Small polynomial Generation function shall be computed by the following or an equivalent sequence of steps;

```
Set dest := 0
Set i := 0
While i <count do
    a. Set ind := rand() % n
    b. If dest [ind] = 0
            i. Set value:=rand()%2
           ii. If value == 0
                  1. value = -1
          iii. dest [ind] =value
    c. Set i:=i + 1
Output dest
```

3.3 Key generation algorithm

Algorithm 1 Random key generation primitive
Component The parameters *n, t, q*
Input Small polynomial Generation Function (*Small polynomial Generation*)
Output Polynomial *F*, Polynomial *h*.
The Random key generation shall be computed by the following or an equivalent sequence of steps;

```
1 Set count := 2n/3+1
2 Call  SmallPolynomialGeneration(count) for computer the polynomial G
3 Set count := 2t
4 Call  SmallPolynomialGeneration(count) for computer the polynomial F
5 Compute the polynomial $f=3F+1$ in $(Z/qz)[X](x^n–x–1)$
6 Compute the polynomial $f^{-1}$= such as $f^{-1}*f=f*f^{-1}=1$ in $(Z/qz)[X](x^n–x–1)$. If $f^{-1}$ not exist go to step 4
7 Compute the polynomial $h = 3gf^{-1}$ in $(Z/qz)[X](x^n–x–1)$
Output F, h
```

The inversion calculation is performed using the extended Euclidean algorithm. When calculating the inversion for the standard NTRU [1], the value *q* (module for polynomial coefficients) was 2048. This made it possible to first calculate the inverse by the module 2, and then use it to calculate the inversion by modules 4, 16, 256, 65536, and then go to module 2048. This made it possible to significantly reduce computing costs compared with the use of the extended Euclidean algorithm [6]. Unfortunately, in NTRUPrime mathematics, *q* is a large prime number, so the authors did not find a way to gradually calculate the inversion.

To reduce the computational complexity of the algorithm, the following techniques were used:

AVX operations were used to perform all operations on polynomials;

The Barrett method was used to calculate the module;

Only positive coefficients were used in the calculation, the transition to the value in the range $[-q/2, q/2]$ was performed after the final calculation of the inversion.

For further optimization, you can simultaneously form Small Polynomials *F*, *G*, but their formation time is no more than 5% relative to the inversion calculation function, so this optimization is not used.

Time characteristics of the key generating function for some NTRUPrime parameters are given in Table 4. The first and 5 columns define the parameter numbers in the Table B Parameters [2]. Numbering from 0. Parameter № 64 (n=739, q=9829) is given for comparison with the data given in [2]. Parameter №74 (n=761, q=4591) is chosen for comparison with the data given in [3].

Table 4

Time characteristics of the key generating function for some NTRUPrime parameters

№	*N*	*Q*	KeyGenerations (tacts)	№	*N*	*Q*	KeyGenerations (tacts)
0	439	6833	17542928	9	479	6089	17317560
1	457	6037	16663192	10	491	6287	17758164
4	467	3911	14472788	15	503	2879	14384316
5	463	6529	17583180	17	523	3331	15206436
6	463	6841	17779208	64	739	9829	29850996
8	479	5689	16777688	74	761	4591	24910804

The maximum time for parameter generation is 38976232, the minimum – 14472788 processor tacts for the last parameter and parameter 4 from the parameter table [2].

4. Algorithms for converting polynomials into an array of bytes and vice versa

4.1 Converting a polynomial $(Z/3z)[X](x^n-x-1)$ into an array of bytes (package)

The algorithm is used to encode a private key.

The coding table for the polynomial coefficients is given (Table 5).

Table 5

Coding table

Code	Coefficients	**Code**	Coefficients	**Code**	Coefficients
00000	-1, -1, -1	**01001**	-1, -1, 0	**10010**	-1, -1, 1
00001	0, -1, -1	**01010**	0, -1, 0	**10011**	0, -1, 1
00010	1, -1, -1	**01011**	1, -1, 0	**10100**	1, -1, 1
00011	-1, 0, -1	**01100**	-1, 0, 0	**10101**	-1, 0, 1
00100	0, 0, -1	**01101**	0, 0, 0	**10110**	0, 0, 1
00101	1, 0, -1	**01110**	1, 0, 0	**10111**	1, 0, 1
00110	-1, 1, -1	**01111**	-1, 1, 0	**11000**	-1, 1, 1
00111	0. 1, -1	**10000**	0. 1, 0	**11001**	0. 1, 1
01000	1, 1, -1	**10001**	1, 1, 0	**11010**	1, 1, 1

To assign 3 polynomial coefficients, it is enough to use 5 bits. Table 5 specifies codes for all variations of the coefficients. The row number of the table specifies the code (5 bits), which encodes 3 consecutive polynomial coefficients, starting with the coefficient with a smaller number.

4.2 Converting a polynomial $(Z/qz)[X](x^n–x–1)$ into an array of bytes and vice versa

The algorithm is used for public keys and encryption results.

The objective is to allocate for the public key and the encryption result minimum of memory to allow storage its on a device with a small memory.

If in the standard q=2048 for all parameters, then for the new algorithm the value q>=48t+3 and the prime number (2t is the number of polynomial non-zero coefficients).

2 methods of packing q.

1 method. Calculate the minimum $Q=2^k>q$. In packing for each polynomial element take k bits.

Advantage – simple packaging-unpacking operation.

Disadvantages:

the public key size can be reduced;

the k value is different for different q, that is, the packaging-unpacking procedures depend on q.

2 method. Set a polynomial as a large number in the q system. That is, to calculate the polynomial value:

$$h_0+h_1*q+h_2*q^2+\ldots+h_{n-1}*q^{n-1}.$$

Advantages:

public key takes a minimum of memory;

packing (calculating the polynomial value – the Gorner scheme)–unpacking (the definition of the number "digits" in a given numbers system) procedures do not depend on q.

q – this is the algorithm parameter, that is, the values q^i for i=2…n–1 can be calculated once, this will greatly accelerate both the packing algorithm and the unpacking algorithm.

Disadvantage: We must use arithmetic of long numbers.

Unpacking – the operation is inversely related to the selected packaging option.

Table 6 shows the values of the public key lengths when using the first and second methods, as well as the time (the number of processor tacts) for key unpacking for some parameters of NTRU Prime. The first column defines the parameter number in the Table B Parameters [2]. Numbering from 0. Parameter № 64 (n=739, q=9829) is given for comparison with the data given in [2]. Parameter №74 (n=761, q=4591) is chosen for comparison with the data given in [3].

Table 6

Public Key. The dependence of the length and time for the unpacking operation depending on the method

№			Lengths (Bits)			Time (tacts)			
	N	*Q*	Len1	Len2	Δ (%)	Pack1	Unpack1	Pack2	Unpack2
0	439	6833	5707	5593	2	3100	3154	1685683	8263155
1	457	6037	5941	5740	4	1356	1380	750852	3706680
4	467	3911	5604	5573	1	1055	1076	701441	3506980
5	463	6529	6019	5868	3	1273	1310	727928	3679286
6	463	6841	6019	5899	2	1270	1288	732257	3692521
8	479	5689	6227	5976	4	1297	1315	767563	3899634
9	479	6089	6227	6022	3	1300	1318	772980	3902594
10	491	6287	6383	6196	3	1358	1376	812741	4107416
15	503	2879	6036	5781	4	1122	1140	780414	3946185
17	523	3331	6036	5886	3	1119	1137	793815	3998625
64	739	9829	10346	9802	6	1902	1920	1880312	9626583
74	761	4591	9893	9258	7	2044	2063	1832085	9494444

Conclusions:

1. The length for the second method is less than the length for the first method, not more than 8 %.

2. The packing-unpacking time for option 1 is much less than the time of the corresponding operation for the second option. The transformation time for the second option is even greater than the time for encryption and decryption operations.

Developers recommendation: use the first option.

5. Polynomials multiplication operation

The multiplication operation is performed during encryption (one operation) and decryption (2 operations). It is this operation that takes most time among the remaining operations, so its optimization is paying much attention.

All multiplication operations are performed for polynomials $(Z/3z)[X](x^n–x–1)$ and $(Z/qz)[X](x^n–x–1)$ with each other. As a result, we obtain a polynomial $(Z/qz)[X](x^n–x–1)$. The multiplication on the polynomial f=3F+1 (decryption operation) can easily be replaced by the multiplication operation by F. Really, $f*h=(3F+1)*h=3*F*h+h$. After calculating $F*h$ multiplication by 3 and adding h are performed very quickly due to the use of AVX operations. As our researches have shown, it is more efficient than calculating directly $f*h$. Therefore, the polynomial multiplication operation with $(Z/3z)[X](x^n–x–1)$ on the polynomial $(Z/qz)[X](x^n–x–1)$ is considered below.

To optimize the multiplication operation, the following was investigated:

—method of specifying the polynomial in $(Z/3z)[X](x^n–x–1)$,

—different multiplication algorithms.

5.1. Method of specifying the polynomial in $(Z/3z)[X](x^n–x–1)$

The polynomial after unpacking has coefficients 0, 1, –1. To specify, you can use an array in which to specify all coefficients, or numbers arrays that have values 1 and –1. When performing a multiplication operation, the most significant coefficients are significant, so the second method of assignment is more accepted.

5.2. Different multiplication algorithms

We have investigated all multiplication methods that are recommended in [2, 3] and other methods. When implementing various methods, minimization of transition operations was performed and the properties of the modern processors cache were taken into account. The possibilities of using parallel computations through the use of AVX operations and multi-core processors were explored.

The multiplication result must be reduced by modulus q (prime number) and by modulus of polynomial $x^n–x–1$. As our studies have shown for the reduction by modulus q to use the Barrett reduction method is most effectively [5].

For reduced by modulus $x^n–x–1$, polynomial $x^n–x–1$ properties are used.

The following is a summary of the various multiplication and calculating modules methods.

5.2.1. Experimental research of multiplication algorithms

The following methods were studied:

- A1 – "School" method – it is provided for comparison and verification of the results correctness;
- A2 – Toom-Cook's algorithm. It is implemented according to the [2, 3] recommendations. Even without interpolation, time characteristics are worse than the rest of the algorithms;
- A3 – FFT, Fast Fourier transform (algorithm with pre-calculations). Time characteristics approximately coincide with the time characteristics for the Toom-Cook's algorithm;
- A4 – Our algorithm that takes into account the special structure of a polynomial with coefficients (0, –1, 1) for which numbers are given, using AVX commands;
- A5 – Algorithm A4, which uses 2 threads.

– A6 – Algorithm A4, which uses 4 threads.

The results of the experimental research for some parameters from NTRU Prime are shown in Table 7. The first column defines the parameter number in the Table B Parameters [2]. Numbering from 0. Parameter № 64 (n=739, q=9829) is given for comparison with the data given in [2]. Parameter №74 (n=761, q=4591) is chosen for comparison with the data given in [3].

Table 7

Polynomials multiplying algorithms. Time indicators

№	*N*	A1	A2	A3	A4	A5	A6
0	439	403796	152191	143679	40179	26588	18436
1	457	451344	157148	147812	37692	26336	18492
4	467	471340	157100	148196	24852	17636	14556
5	463	459120	157064	148352	46188	27444	20008
6	463	471340	157100	148196	24852	17636	14556
8	479	494224	157144	148020	36664	22468	18096
9	479	494156	157204	148228	39408	26168	18236
10	491	519168	157096	148556	41048	25784	19084
15	503	545388	157168	148120	19464	16376	13952
17	523	545436	157192	148056	22608	18176	14704
64	**739**	**1162612**	**157136**	**148944**	**89676**	**52344**	**32400**
74	**761**	**1229540**	**157204**	**148852**	**38560**	**27264**	**20148**

Conclusions on the multiplication methods:

1. The use of complex algorithms that do not take into account the special structure of the polynomial with coefficients (–1, 0, 1) makes no sense.

2. The polynomial with coefficients (–1, 0, 1) is better to be specified using non-zero elements indices.

3. The use of threads in the case of module performance on the multi-core processor makes a sense.

4. According to [2] for the parameters (p=739, q=9829) the number of tacts for the multiplication operation is 51488, we have 32400.

5. According to [3] for the parameters (p=761, q=4591) the number of tacts for the multiplication operation is 28682, we have 20148.

After completing the multiplication operation, we obtain the polynomial, whose coefficients do not exceed $q*n$, and the degree of the polynomial is $2n–1$. This polynomial must first be reduced by modulus $x^n–x–1$, after that we obtain the polynomial of degree n, and then each of n coefficients are reduced by modulus q.

5.2.2. Reduced by modulus $x^n–x–1$

For reduction by modulus $x^n–x–1$ it is sufficient to polynomial coefficients with indices $0…n–1$ to add (subtract) corresponding coefficients with indices $n...2n–2$.

To simultaneously reduction the coefficients block using AVX operations.

5.2.3. Reduced by modulus q. Barrett reduction

To accelerate the method, the constants, which depend only on the values of n, q, are computed one time when setting parameters.

Barrett bk, bc constants pre-calculation:

bk is chosen such that $2^{bk}>p*q$

$bc=2^{bk}/q$ is calculated.

For each polynomial coefficient h_i, it must perform the following calculations:

$$h_i=h_i–((h_i*bc)>>bk)*q.$$

To simultaneously reduction by modulus q the coefficients block using AVX operations.

6. Encryption and decryption

6.1. Encryption

The encryption algorithm coincides with Algorithm 23 (ANSI X 9.98 [1]). Next, the notation is used from Algorithm 23.

6.1.1. Encryption algorithm optimization

1. For strings *M* and *sData*, intersecting memory is used. This allows you to reduce the amount of memory required by the message length and reduce the time it takes to copy the string for encryption (Step 5 and Step 9).
2. To accelerate the formation of *Mtrin* (Step 8), the bit string is processed in portions of 3 bytes, which allows you to immediately get 8 polynomial coefficients. The case is handled correctly when the length of the string is not multiple 3.
3. To exclude the need to convert a public key into a byte string (Step 9), it is stored in the container in the form of a byte string and in the form of a polynomial.
4. For blinding polynomial generation (Step 10) a BPGM method is used, which optimization will be described below.
5. To calculate *r*h* (Step 11), the multiplication function is used for one or multi-core processor according to the execution environment. The maximum number of cores that the function uses is 4.
6. Step 12 and Step 13 are executed as one step.
7. The polynomial generation to mask *mask* (Step 14, MGF).
8. Steps 15–18 are executed as one step in which the modules are formed and the numbers of values –1, 0, 1 are calculated, their correctness is checked and the ciphertext value is calculated.
9. The result is the bytes array that we obtain by converting a polynomial into bytes array according to the package algorithm.

6.1.2. Optimization of the blinding polynomial calculation algorithm

Blinding polynomial calculations are performed according to the Blinding Polynomial Generation Method (BPGM) – Algorithm 18 [1].

The algorithm uses the index generation function (IGF) by which the bit string (IGF state *s*) creates, with the length *minCallsR*Hlen*.

6.1.3. Formation of IGF state *s*

We consider 3 options for creating bit string.

For option 1, use of the hash function as proposed in the standard [1]. To optimize the implementation of the *s* creation function, a constant part is formed that is used at each step of the hash calculation. When forming a hash, AVX commands are used.

For option 2, when implementing the *s* creation function, the initial string and its length are defined as for option 1, but instead of multiple recalling of the hash function, the multiple encryption function call for the SALSA-2.0 algorithm is used. For the next step of encrypting, the result for the previous step is selected. The number of steps compared to the Algorithm for option 1 is reduced due to the fact that the length of the initial state is longer than the length of the hash.

For option 3, instead of the hash function, the encryption function for the SNOW-2.0 algorithm is used (see option 2).

6.1.4. Calculation of coefficients

To optimize the calculation of the polynomial coefficients in the first step, a completely array of coefficients is formed. At the second step, the possibility of their application is checked. If necessary, the bit string is expanded.

6.1.5. MGF algorithm optimization

The algorithm uses the *minCallMask* parameter to generate a bit string. The bit string formation optimization is made due to the fact that the constant part is used at each step of the hash calculation is calculated only once. When forming a hash, AVX commands are used. For fast conversion of a byte string in a polynomial, a pre-computed table is used, the entry point of which is byte and each row includes 5 coefficients.

6.1.6. Parallel computing using

In the encryption algorithm, 2 branches are executed in parallel.

The first branch includes Steps 4-8 of the encryption algorithm (Algorithm 23 [1]).

The second branch includes Steps 9-10 (Algorithm 23 [1]).

To implement parallel branches, Open MP is used.

6.2. Decryption

The decryption algorithm is executed according to Algorithm 23 (ANSI X 9.98 [1]). Next, the notation is used from Algorithm 23 [1].

6.2.1. Decryption algorithm optimization

1. Steps 1-3 are combined in one step. The multiplication algorithm is used to multiply polynomials. The number of non-zero elements of received polynomial is counted simultaneously with its formation.
2. Steps 4-6 are combined in one step, that allows to form in one cycle a byte row to calculate *c0R4*.
3. Step 7 optimization.
4. To optimize Step 8, the cycle is deployed to simultaneously receive 4 bytes of string.
5. Steps 9, 10, 11 are combined in one step. That allowed in one cycle to form a byte string.
6. Step 12 of the algorithm actually determines the message after the decryption. Further, the speculative execution of the code that uses these data may be continued. The following steps can be performed in parallel with the use of the obtained data. If in result of additional checks will be obtained a negative result, the code execution after the use of decrypted data should be determined to be invalid. In case of successful additional verification, the executed code is accepted as valid. Additional checks include Steps 13-17.
7. For Step 13, you do not need to convert the public key into a byte string, it is stored in this format.
8. Step 14 optimization to form a blinding polynomial.
9. Step 15 polynomials multiplying.

6.3. Time characteristics of encryption and decryption functions

The results of the experimental research for some parameters from NTRU Prime are shown in Table 8. The first column defines the parameter number in the Table B Parameters [2]. Numbering from 0. Parameter № 64 (n=739, q=9829) is given for comparison with the data given in [2]. Parameter №74 (n=761, q=4591) is chosen for comparison with the data given in [3].

For encryption and decryption operations, there are 3 modes for creating a random string to create a blinding polynomial (hash, salsa2.0, snow2.0). For both modes, we get the best results for the last option.

The last 2 columns determine the time it takes to decrypt and verify the decryption correctness. In the case of parallel execution of the verification operation with other encryption-decryption operations, you can balance the time required for encryption and decryption.

The last row in the table specifies the results obtained for the parameters specified in [3]. These parameters correspond to a cryptographic stability of more than 200.

Authors [3] received the results:

Encryption: 59600 and decryption 97452 cycles respectively.

Table 8

Time characteristics of encryption and decryption functions

№	N	Q	Encrypt			Decrypt			Decrypt	
			Hash	Salsa2.0	Snow2.0	Hash	Salsa2.0	Snow2.0	Decrypt	Check
0	439	6833	56364	40456	35408	81356	64280	60016	29048	28508
1	457	6037	62712	38200	33712	87948	61448	56868	30500	28496
4	467	3911	52340	32016	29252	72760	51528	48532	26908	23320
5	463	6529	64336	39124	35288	90484	63256	59072	32352	29220
6	463	6841	66240	40628	36268	94884	66448	61464	32336	30884
8	479	5689	62800	38396	34144	87748	61556	57320	31000	28360
9	479	6089	64444	39072	34796	90624	62596	58352	32168	28948
10	491	6287	67464	41304	36188	94220	65984	61500	31368	30600
15	503	2879	50460	30536	27844	87748	61556	47116	31000	28360
17	523	3331	54192	33112	30812	90624	62596	51224	32168	28948
64	739	9829	104916	61597	53876	145516	100152	93224	48636	46488
74	761	4591	73456	44120	40032	100140	71480	67728	37580	30920

In the case of the hash using, our encryption operation implementation loses the specified ones in the paper by 25%, and the decryption operation – 5%. When using encryption algorithms, our implementation wins 30% for encryption algorithm and 29% for decryption algorithm.

In [4] – one of the algorithms submitted to the competition (Kyber), the following performance data after optimization using AVX2 are given:

Encryption – 119652;

Decryption – 125736.

Compared to our results for the best option, we get the winning: 65% for encryption and 44% for decryption.

Conclusions

In view of the above, the following conclusions can be made.

1. In the cryptosystem «NTRU Prime IIT Ukraine» as the main cryptographic transformation, as in NTRU Prime, unlike NTRU, the transformation is used in the finite field. The above makes it impossible to conduct a series of potential attacks regarding the cryptographic system «NTRU Prime IIT Ukraine» and eliminates the potential weaknesses present in the NTRU cryptosystem. They are mainly related to the existence of non-trivial subfields or factor rings of the factor (truncated) polynomials ring.

2. In the cryptosystem «NTRU Prime IIT Ukraine» polynomials F and r are arbitrary t-small, they have $2t$ non-zero coefficients (+1, –1), whereas in NTRU, each of these polynomials has exactly t nonzero coefficients equal to 1 and –1 respectively. The same is true for the polynomial g used in the cryptosystem «NTRU Prime IIT Ukraine», which is an arbitrary small polynomial with $2t$ nonzero coefficients (+1, –1). Specified allows to expand the size of the key space in comparison with NTRU without losing the efficiency of algorithms implementation for the keys formation and implementation of encryption and decryption algorithms.

3. During optimization great attention was paid to multiplication operation, as it is the most time consuming. Usage of complex multiplying algorithms, which don't take into account special polynomial structure with coefficients (–1, 0, 1) doesn't make a sense. The polynomial with coefficients (–1, 0, 1) is better to be specified using non-zero elements indices. The use of threads in case of reduction by modulus on the multi-core processor makes a sense. Usage of AVX2 operations for reduction by modulus polynomial $x^n - x - 1$ and prime q and for Barrett algorithm for reduction by modulus q are effective and accelerates multiplication speed.

4. Three algorithms of blinding polynomial formation were studied (hash, salsa2.0, snow2.0), the best time rates were obtained for Snow 2.0.

References:

1. American National Standard for Financial Services Lattice-Based Polynomial Public Key Establishment Algorithm for the Financial Services Industry ANSI X9.98–2010, 2010. 284 p.

2. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, Christine van Vredendaal: NTRU Prime. Access mode: https://ntruprime.cr.yp.to/ntruprime-20160511.pdf.

3. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, Christine van Vredendaal NTRU Prime: reducing attack surface at low cost. Access mode: https://eprint.iacr.org/2016/461.pdf.

4. Joppe Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck CRYSTALS Kyber: a CCA-secure module-lattice-based KEM. Access mode: https://eprint.iacr.org/2017/634.pdf.

5. P. Barrett Communications, Authentication and Security Using Public Key Encryption A Design for Implementation. Master's thesis, Oxford University, September 1984.

6. Kachko E. G. Investigation of inversion calculation methods in the NTRU algorithm / E. G. Kachko, D. S. Balagura, K. A. Pogrebniak, Yu. I. Gorbenko // Radiotekhnika. 2012. Issue 171. P. 58–63. (in Russian).

7. Post-Quantum Cryptography. [Electronic resource] Access mode: https://csrc.nist.gov/projects/post-quantum-cryptography.

8. Kachko O. G. The optimization of NTRU-like algorithm for asymmetric encryption with "inconvenient parameters" / O. G. Kachko, L. V Makutonina, O. S. Akolzina // Mathematical and computer modeling. Series: Engineering, 15 (2017), 79–85. (in Ukrainian).

9. Kachko O. Asymmetric encryption algorithm optimization based on using NTRU Prime mathematics / O. Kachko, Yu. Gorbenko, M. Yesina, O. Akolzina // Radiotekhnika. 2017. Issue 191. P. 5-10.

Акціонерне товариство
«Інститут інформаційних технологій»,
Харківський національний
університет радіоелектроніки,
Харківський національний
університет імені В.Н.Каразіна

Надійшла до редколегії 11.02.2018