МЕТОДЫ И МЕХАНИЗМЫ КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ

UDC 004.056.55

O. KACHKO, Yu. GORBENKO, M. YESINA, O. AKOLZINA

ASYMMETRIC ENCRYPTION ALGORITHM OPTIMIZATION BASED ON USING NTRU PRIME MATHEMATICS

Introduction

The development in quantum computer creation caused the need to search for quantum-resistant cryptographic algorithms and requirements formation for them. So NIST at the fall of 2017 announced a request for post-quantum algorithms search, including algorithms for asymmetric encryption [2]. It is known, for practical application, algorithms have to satisfy the requirements of resistance, performance and should be lightweight. During the work, optimization of a perspective post-quantum encryption NTRU-like algorithm was carried out. The encryption scheme from standard ANSI X9.98-2010 [1] and NTRU Prime parameters [3] were used in implementation.

1. NTRU Prime cryptosystem

Parameters for key generation, encryption and decryption, their appointment and formulas are specified in table 1 below. Then we describe formulas for keys generation, encryption and decryption according to [3].

Table 1

Denotation	Appointment	Formula
n	Polynomial order. Determines the number of its coefficients. A prime number for which the polynomial $x^n - x - 1$ is irreducible.	$n \geq \max\{3, 2t\}$
R	Field of polynomials $Z[x]$ with modulus $x^n - x - 1$.	$Z[x]/(x^n - x - 1)$
$R/3$	Field of polynomials $(Z/q)[x]$ with modulus $x^n - x - 1$.	$(Z/3)[x]/(x^n - x - 1)$
R/q	Field of polynomials $(Z/q)[x]$ with modulus $x^n - x - 1$.	$(Z/q)[x]/(x^n - x - 1)$
p	Smaller modulus.	$p = 3$
q	Bigger modulus, a prime number, by which all coefficients of a polynomial R/q are reduced.	$q \geq 48t + 3$
t	The natural number, determines the number of non-zero elements of a polynomial.	$t \geq 1$
k	Security strength (level).	
m	Secret message. The number of 0, 1 and -1, is bigger or equal t.	$m \in R/3$
e	Encrypted message.	$e \in R/q$
g	Random polynomial, that has $2N/3$ non-zero elements. The number of 1 and -1 is not necessarily equal. The secret parameter used to calculate the public key.	$g \in R/3$
f	Random t-small element (polynomial) is a secret.	$f = (1 + 3F) \bmod q$ $f \in R/q$
F	Random t-small element (polynomial) that defines a private key	$F \in R/3$
h	The sender's public key. Invertible element in R/q. The length of h is equal to $n\lceil \log_2 q \rceil$.	$h = 3g/f \in R/q$
r	Blinding polynomial, random t-small element.	$r \in R/3$
b	Random sequence (salt) that pads the message (Length of b is defined by security strength).	

Keys generation. By definition secret key is a polynomial f, $f = (1+3F) \bmod q$, where $F \in R/3$, the number of non-zero elements $\| F \|_1 = 2t$, and corresponding public key is polynomial $h = 3g / f \in R/q$, where $g \in R/3$.

For any secret key f and corresponding public key h define *encryption* and *decryption* functions E_h and D_f:

$$e = E_h(m,r) = (m + rh) \bmod q,\ m,r \in R/3,\ \| r \|_1 = 2t, \quad (1)$$

$$D_f(e) = (ef \,(\bmod q) \bmod 3,\ e \in R/q. \quad (2)$$

2. Experimental research of multiplying algorithms

Encryption and decryption algorithms use the multiplication functions of polynomials that have big computational complexity, therefore we investigated various methods of calculating the product first of all.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 [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 AVX2 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 2 The first column defines the parameter number in the Table B Parameters [3].

Table 2

N	A1	A2	A3	A4	A5	A6
439	403796	152191	143679	40179	26588	18436
457	451344	157148	147812	37692	26336	18492
461	459088	157196	148476	46116	29808	20212
461	459120	157064	148352	46188	27444	20008
467	471340	157100	148196	24852	17636	14556
463	462736	157196	148268	40900	24724	18864
463	462828	156972	148608	43312	28768	19700
463	462676	157164	148460	46788	30672	20220
479	494224	157144	148020	36664	22468	18096
479	494156	157204	148228	39408	26168	18236
491	519168	157096	148556	41048	25784	19084

Conclusions on the multiplication methods:

1. The usage of complex algorithms (A2, A3) that do not take into account 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 (A4, A5, A6).

3. The use of threads in case of modulus reduction on the multicore processor makes sense.

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

3 Reducing by modulus $x^n - x - 1$ optimization

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

To simultaneously reduction the coefficients block using AVX2 operations.

4 Reducing by the modulus q – Barrett reduction optimization

To accelerate the method, the constants, which depend only on the values of n, q, are computed ones when setting parameters [9].

Barrett bk, bc constants pre-calculation:

$$bk \text{ is chosen such that } 2^{bk} > p*q; \quad (3)$$

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

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

$$h_i = h_i - ((h_i * bc) >> bk) * q. \quad (4)$$

To simultaneously reduction by modulus q the coefficients block using AVX2 operations.

5. Optimization of Blinding polynomial calculation algorithm

Blinding polynomial calculation is 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 $minCallsR * Hlen$ (see Algorithm 20 [1]). When implementing the creation function s, a constant part is formed that is used at each step of the calculation of the hash. When forming a hash, AVX2 commands are used.

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

5.1. Optimization of IGF state *s* formation

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 value calculation. When forming a hash value, AVX2 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-20 algorithm is used [7]. 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 value.

For option 3, instead of the function for algorithm SALSA-20, the encryption function for the SNOW-20 algorithm is used [4].

5.2. Coefficients calculation optimization

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.

Blinding polynomial is used for data encryption and decryption. The effect of various methods of blinding polynomial formation is shown in table 3.

6. MGF algorithm optimization

The algorithm uses the $minCallMask$ parameter (see Algorithm 19 [1]) 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, AVX2 commands are

used. For fast conversion of a byte string into a polynomial, a pre-computed table is used, the entry point of which is byte and each row includes 5 coefficients.

7. Encryption algorithm optimization

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, the Open MP parallelization standard is used [8].

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

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 above (see 5).

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 of Algorithm 19 [1]).

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 an array of bytes that we obtain by converting a polynomial into an array of bytes according to the package algorithm. To optimize the latter, separate algorithms are developed, depending on the bit length q.

8. Decryption algorithm optimization

The decryption algorithm is executed according to Algorithm 24 [1]. Next, the notations are used from Algorithm 24.

1. Steps 1-3 are combined in one step. The multiplication algorithm A4–A6 is used to multiply polynomials depending on the number of processor cores. 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 (see paragraph 6).

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 (see paragraph 5).

9. Step 15 polynomials multiplying.

9. Time rates of encryption and decryption functions

The results of the experimental research for some parameters from NTRU Prime are shown in Table 3. The first column defines the parameter number in the Table B Parameters [3]. Number ring from 0. Parameter № 64 (n=739, q=9829) is given for comparison with the data given in [3]. Parameter №74 (n=761, q=4591) is chosen for comparison with the data given in [5].

Table 3

№	*N*	*Q*	Encrypt			Decrypt			Decrypt	
			Hash	Salsa20	Snow20	Hash	Salsa20	Snow20	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

For encryption and decryption operations, there are 3 modes for creating a random string to create a blinding polynomial (hash, salsa20, snow20), see paragraph 5. For both functions, we get the best results for the last mode.

The last 2 columns determine the time taken 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 [5]. These parameters correspond to a cryptographic stability of more than 200.

Authors [5] received the results:

Encryption: 59600 and decryption 97452 cycles respectively.

In the case of the hash value using, our encryption operation implementation loses the specified ones in the by 25%, and the decryption operation – 5%. When using SNOW 2.0, our implementation wins 30% for encryption algorithm and 29% for decryption algorithm.

In [6] – 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

According to the results of the work we can made the following conclusions:

1. 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 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 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.

2. Blinding polynomial generation, coefficient calculation, blinding polynomial generation and index generation function optimizations were also made.

3. Encryption and decryption algorithms were optimized due to the parallel computing.

4. Three algorithms of blinding polynomial formation were studied (hash, salsa20, snow20), the best time rates were obtained for Snow 2.0.

5. In case of usage encryption algorithms our implementation wins 30 % for encryption algorithm and 29 % for decryption algorithm.

6. Compared to the data given in [6], we have got the winning: 65 % for encryption and 44% for decryption.

References: 1. American National Standard X9.98-2010. Lattice-based polynomial public key encryption algorithm, Part 1: key establishment, Part 2: data encryption. – 2010. 2. Electronic resource: https://csrc.nist.gov/projects/post-quantum-cryptography. 3. *Bernstein D.J., Chuengsatiansup Ch., Lange T., van Vredendaal Ch.* NTRU Prime // Cryptology ePrint Archive: https://ntruprime.cr.yp.to/ntruprime-20160511.pdf. 4. *Patrik Ekdahl, Thomas Johansson*. A New Version of the Stream Cipher SNOW // SAC 2002: Selected Areas in Cryptography pp 47-61. 5. *Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, Christine van redendaal*. NTRU Prime: reducing attack surface at low cost // Cryptology ePrint Archive: https://eprint.iacr.org/2016/461. 6. *Joppe Bos, Leo Ducas, Eike Kiltz*. CRYSTALS – Kyber: a CCA-secure modulus-lattice-based KEM // https://eprint.iacr.org/2017/634. 7. *Daniel J. Bernstein.* Salsa20 design // https://cr.yp.to/snuffle/design.pdf 8. Electronic resource http://www.openmp.org/. 9. P D Barrett, "Communications Authentication and Security using Public Key Encryption – A Design for Implementation." (Oxford University Programming Research Group MSc Thesis (1984).

Харьковский национальный
университет радиоэлектроники,
Акционерное общество
«Институт информационных технологий»,
Харьковский национальный
университет имени В.Н. Каразина

Поступила в редколлегию 10.10.2017