Probabilistic Block Cipher

Nikita Zbitnev, Dmitry Shishlyannikov, Dmitry Gridin
JetBrains Research, Novosibirsk, Russia
Novosibirsk State University, Novosibirsk, Russia
Email: nikita.zbitnev.a@gmail.com, dmitry.shishlyannikov.a@gmail.com, dmitry.gridin.v@gmail.com

Abstract—**This paper is devoted to the description of a new block cipher that will be applicable in the post-quantum era and will not need a lot of resources. The main advantages: probabilistic encryption, the cipher block chaining mode, the ability to transfer to distributed systems. All this combined with the use of PRNG, working on the Cremona transformations, has significantly increased the cryptographic strength and increased the scope of this encryption.**

I. Introduction

NIST recommendations for block encryption consist in increasing the size of the key. We believe that instead of increasing the key, it is necessary to build completely probabilistic encryption in such a way that the output file size would be unpredictable, that is, to make the output file size unpredictable, even if the key is the same. In addition, it is necessary to build a pseudo-random number generator that receives parameters depending on the time, or other parameters of the computer. To achieve this, the finite rings - adapted Cremona transformation is used. The case is that this kind of transformation in real numbers is in use for fractals constructing.

II. Algorithm Description

A. *Formulation of the problem*

It is necessary to encrypt a block of text B, which has n symbols of b-bit size each with secret key.

B. *Key*

The key is used to generate matrices and to fill in the tables of frequencies in Huffman coding. The assumed key size is:

$$l = 2 * b * n \qquad (1)$$

To extract all the necessary data from the key we would use a PRNG (hereinafter the Generator), which is described in more detail below.

C. *Preparation phase*

1) Module Choice

Let us choose a set of prime integers $M_1 < M_2 < \ldots < M_m$ such as:

$$M = M_1 \cdot M_2 \cdot \ldots \cdot M_m \in [2^b, 2^{b+1}] \qquad (2)$$

It is necessary for having unique decomposition any symbol of b-bit size in remainders by these numbers [2].

2) Generation of Permutations

To provide nonlinear encryption it is necessary to obtain permutations for arithmetic operations for each module from the secret key.

To the module M_i a permutation is a set of prime integers of the type: $\langle a_0, a_1, a_2, \ldots, a_{M_i - 1}\rangle$, where $a_i \in [1, M_i - 1]$ and $a_i \neq a_j$, $i \neq j$. This set is obtained by Generator by the induction algorithm [1].

The permutation changes the addition and multiplication tables for each module [1].

3) Generation of Matrices

For each module invertible matrices $A_1, A_2, \ldots, A_m$ of $n \times n$ size are generated [1].

4) Generation of Huffman Table

In order to increase the cryptographic strength of the algorithm at one of the stages of encryption, we use the Huffman algorithm [1], [3], in which the occurrence frequency for each symbol are obtained by Generator. This choice may be justified by the fact that the combination of Huffman coding and insertion of fake symbols which are described below leads to the changes in the size and content of the encrypted message at every other ciphering though the key is not changed. This helps to resist entropic methods of cracking and known-plaintext attacks [4].

D. *Move-To-Front*

In order to add the block chaining mode - the phenomenon in which changing the block of source text leads changing not only the corresponding encrypted block, but all following blocks, the technology Move-To-Front [3] is used.

Let us present the Huffman table in the form of two arrays: one contains symbols, which are need to be encrypted, a dictionary, and the other is bit representations. Initially, the dictionary will be represented as a sequence from 0 to M_m. The encoding consists of sequentially traversing all the characters of the encrypted vectors. The symbol is searched for in the dictionary, and its bit representation is written to the output stream. After that, the symbol is removed from its position and inserted into the beginning of the dictionary. The second array remains unchanged.

This work was supported by JetBrains Research

For decoding it is necessary to do similar actions with encoded text, in this case the initial state of the dictionary must be identical to the initial state of the encoding dictionary.

E. Fake Symbols

As mentioned above, the algorithm has a stronger cryptography if encrypted data look different at every other ciphering with the same key. We use fake symbols, which change the look of the encrypted message for this purpose.

Obviously, with a symbol size of b, the encrypted characters can not have a value greater than 2^b. Also, according to the Chinese remainder theory, from the remainders of the division of an integer we can collect a number which is smaller than the multiplication of all modules M [2]. Then, if a number λ is added into a block would satisfy following condition $2^b < \lambda < M$ it will be clear that the number is not a symbol of the source file when the block is deciphered and collected.

Each time a new block is read, a decision, whether to insert a fake symbol, made by Generator. The symbol λ itself is also obtained by means of the Generator [1].

F. Ciphering

We will choose modules: $M_1, \dots, M_m$. Secret key is used to generate permutations, matrices $A_1, A_2, \dots, A_m$ for each module and a Huffman table. Read the block B of size $n * b$ and ciphered as follows:

1. The block $B = q_1, q_2, \dots, q_n$ is represented in the form of a vector $\vec{B} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{pmatrix}$. The coordinates of the vector is remainders of $M_1, \dots, M_m$. We would have vectors $v_1 = \begin{pmatrix} v_{1_1} \\ v_{1_2} \\ \vdots \\ v_{1_n} \end{pmatrix}, v_2 = \begin{pmatrix} v_{2_1} \\ v_{2_2} \\ \vdots \\ v_{2_n} \end{pmatrix}, \dots, v_m = \begin{pmatrix} v_{m_1} \\ v_{m_2} \\ \vdots \\ v_{m_n} \end{pmatrix}$, where $v_{i_j} = q_j \bmod M_i$.
2. Matrices are multiplied by the relevant vectors (in every module), all operation we do by tables of multiplication and addition generated previously, vectors of such form are obtained
$$v_i' = A_i \cdot v_i, \tag{3}$$
where $i = 1..m$.
3. The resulting vectors are recorded a special order in accordance with the Huffman table. The order is defined by the sequence of numbers $P = \{p_1, p_2, \dots, p_m\}$, where $p_i \in [1, m]$ – position of this module, $p_i \neq p_j$ if $i \neq j$. So, the vectors are recorded as a sequence: $v_{p_1}, v_{p_2}, \dots, v_{p_m}$.

When moving from block to block, matrices, Huffman tables and the order of record of the encrypted vectors are changed. Matrices are changed through string/column exchange operations or transpositions to save their invertibility.

The order of record of the encrypted vectors is changed through the right circular shift of P to the right to number obtained from the Generator.

The Huffman table is modified according to the Move-To-Front algorithm described above, based on the result of vectors multiplication.

G. Deciphering

We will choose modules: $M_1, \dots, M_m$. Secret key is used to generate permutations, matrices $A_1, A_2, \dots, A_m$ for each module and a Huffman table. Inverse matrices $A_1^{-1}, \dots, A_m^{-1}$, all operation we do by tables of multiplication and addition generated previously.

1. We read m encrypted vectors $v_1', \dots, v_m'$, decoding each symbol in accordance with the Huffman table and change the Huffman table according to the MTF dictionary in the opposite direction.
2. An invertible matrix is multiplied by the relevant vector, all operation we do by tables of multiplication and addition generated previously: $A_i^{-1} \times v_i'$, $i = 1, \dots, m$. Result of this operation is deciphered vectors $v_1, \dots, v_m$.
3. According to the Chinese remainder theory the obtained vectors are collected into the block B. This is a deciphered block. If it has a symbol the numerical value of which is more than 2^b, the symbol is assumed as fake and is thrown out of the block.

When moving from block to block, the changes in the reading order and in Huffman tables are done similarly to the ciphering procedure. In case of invertible matrices, string permutations are replaced by column permutations. Transposition remains unchanged.

III. PRNG and Cremona transformation

A fairly large part of the encryption job is tied to a pseudo-random number generator. In order to exclude the possibility of such an important detail to become a weak point, we decided to take the Cremona transforms [5], [6] as the basis of the principle of the generator's operation. The Cremona transformation is an invertible polynomial mapping from the vector space R^n onto itself, which is given by polynomial functions:

$$\begin{gathered} h_1(x_1, x_2, \dots, x_n), \\ h_2(x_1, x_2, \dots, x_n), \\ \dots \\ h_{n-1}(x_1, x_2, \dots, x_n), \\ h_n(x_1, x_2, \dots, x_n); \end{gathered}$$

invertibility means that equations system:

$$\begin{gathered} h_1(x_1, x_2, \dots, x_n) = a_1, \\ h_2(x_1, x_2, \dots, x_n) = a_2, \\ \dots \\ h_{n-1}(x_1, x_2, \dots, x_n) = a_{n-1}, \\ h_n(x_1, x_2, \dots, x_n) = a_n; \end{gathered} \tag{4}$$

is solvable for any right-hand side.

We fix some module m such that raising to the power $k_1, \dots, k_n$ is a one-to-one mapping modulo m. These degrees can be the same.

We will introduce the mapping:

$$\begin{gathered} p_1 = (u_1)^{k_1} + f_1(u_2, u_3, \dots, u_n), \\ p_2 = (u_2)^{k_2} + f_2(u_3, \dots, u_n), \\ \dots \\ p_{n-1} = (u_{n-1})^{k_{n-1}} + f_{n-1}(u_n), \\ p_n = (u_n)^{k_n}; \end{gathered} \tag{5}$$

Here $f_1(u_2, u_3, \dots, u_n)$, $f_2(u_3, \dots, u_n)$, …, $f_{n-1}(u_n)$ are arbitrary polynomials in the indicated variables. We call this mapping an upper-triangular Cremona mapping and denote this mapping **F**.

The lower-triangular Cremona map is defined similarly. Let the raising to the power $s_1, s_2, \dots, s_n$ to be a one-to-one mapping modulo m. Lower-triangular mapping Cremona:

$$\begin{gathered} q_1 = (v_1)^{s_1}, \\ q_2 = (v_2)^{s_2} + g_1(v_1), \\ \dots \\ q_{n-1} = (v_{n-1})^{s_{n-1}} + g_{n-2}(v_1, v_2, \dots, v_{n-2}), \\ q_n = (v_n)^{s_n} + g_{n-1}(v_1, v_2, \dots, v_{n-2}, v_{n-1}); \end{gathered} \tag{6}$$

we will denote this map **G**.

It is obvious, that both maps are invertible.

We can take a superposition with linear invertible mappings that are represented by $n \times n$ matrices, and these matrices are invertible modulo m. In addition, we can use invertible affine mappings that are determined by an invertible matrix and the vector **w**. Let us give a simple example. Consider the case:

$$n = 2,$$
$$m = 11;$$

Upper-triangular transformation:

$$p_1 = (u_1)^3 + 2 * (u_2)^2,$$
$$p_2 = (u_2)^5;$$

Lower-triangular transformation:

$$q_1 = p_1,$$
$$q_2 = p_2 + (p_1)^2;$$

Superposition:

$$\begin{gathered} q_1 = (u_1)^3 + 2 * (u_2)^2, \\ q_2 = (u_2)^5 + [(u_1)^3 + 2 * (u_2)^2]^2 = \\ = (u_2)^5 + (u_1)^6 + 4 * (u_1)^3 * (u_2)^2 + 4 * (u_2)^4; \end{gathered}$$

Suppose given a matrix

$$\mathbf{A} = \begin{pmatrix} 3 & 5 \\ 10 & 2 \end{pmatrix}$$

Consider the superposition of **GAF**.

AF:

$$\begin{pmatrix} 3 & 5 \\ 10 & 2 \end{pmatrix} \begin{pmatrix} (u_1)^3 + 2(u_2)^2 \\ (u_2)^5 \end{pmatrix} = \\ = \begin{pmatrix} 3(u_1)^3 + 6(u_2)^2 + 5(u_2)^5 \\ 10(u_1)^3 + 9(u_2)^2 + 2(u_2)^5 \end{pmatrix}$$

GAF:

$$\begin{gathered} 3(u_1)^3 + 6(u_2)^2 + 5(u_2)^5 \\ 10(u_1)^3 + 9(u_2)^2 + 2(u_2)^5 + \\ +[3(u_1)^3 + 6(u_2)^2 + 5(u_2)^5]^2 \end{gathered}$$

This composition can be used any number of times, thus increasing the degree of polynomials. In this case, if the degrees of the transformations p_i and q_i are less than or equal to 2, then the possibility to restore the initial data will be preserved. That as a result will make it possible to infinitely complicate the relationship between the initial data. The best application of this fact can be found in the generation of public keys and the exchange of keys.

By increasing the dimensionality of the matrix, we can also control the length of the resulting vector, which makes the algorithm easily scalable.

Cremona transformations can be used not only to generate a sequence of polynomials of any degree, but also to generate pseudo-random vectors. Consider the following example:

$$\begin{aligned} &m = 11, \\ &x = u^7 + v^4 + 5, \\ &y = v^5, \\ &p = x, \\ &q = x^2 + 2y, \\ &x_1 = 10, \\ &y_1 = 8, \\ &x_{i+1} = {x_i}^7 + y_i^4 + 5, \\ &y_{i+1} = ({x_i}^7 + y_i^4 + 5)^2 + 2 * y_i^5; \end{aligned}$$

$\binom{10}{8}$; $\binom{4}{9}$; $\binom{0}{5}$; $\binom{10}{5}$; $\binom{9}{5}$; $\binom{3}{6}$; $\binom{8}{9}$; $\binom{8}{6}$; $\binom{1}{10}$; $\binom{3}{1}$; $\binom{0}{6}$; $\binom{10}{1}$; $\binom{1}{0}$; $\binom{2}{5}$; $\binom{6}{0}$; $\binom{9}{1}$; $\binom{6}{7}$; $\binom{1}{3}$; $\binom{6}{6}$; $\binom{7}{2}$; $\binom{1}{2}$; $\binom{7}{7}$; $\binom{10}{10}$; $\binom{1}{7}$; $\binom{5}{2}$; $\binom{9}{9}$; $\binom{10}{3}$; $\binom{4}{2}$; $\binom{0}{1}$; $\binom{2}{7}$;…

This sequence has a period of about 30 operations, but if you change the input data and the transformation degree, we can achieve a lager period. This fact has yet to be explored in more detail. Since for the generator we do not need the invertibility of operations, the degrees of the transformations p and q can be arbitrary. However, the largest period is observed in powers that can be uniquely inverted for most field elements.

Generators of this kind will be useful if there will be necessity for generating a set of numbers for further work, for example, determining the insertion of fake symbols into a sequence of blocks at once, and not on each block separately.

IV. Cryptanalysis

A. Probabilistic encryption

The basis of probabilistic encryption in our algorithm is fake symbols. These symbols can easily be added to the source text, using any PRNG to determine the character and its position, and it is also easy to detect them while decrypting. The algorithm also does not impose any restrictions on the number of fake characters in one block, which allows you to change their number depending on our needs. Therefore, on small texts you can use not only fake symbols, but also the fake blocks, distributing useful information among random places of ciphertext.

The number of multipliers, the source text block divided by, also affects the size of the ciphertext. In combination with the Huffman table, this fact allows you to fully control the size of the encrypted text, that means you can both increase the size of the output text, and reduce.

Thus, the size of the resulting text becomes unpredictable, which forces the attacker to pick up the initial parameters before the attack begins, that is completely restore the algorithm. So, the problem of hacking is a full search of options. Even on the same source text and the same key, the ciphertext will be different. This is ensured by the use of a PRNG, based on some non-persistent parameter, for example, the time, the processor clock rate. As a source of entropy, any time-dependent variable will do.

B. Avalanche effect and block chaining mode

If any bit in any block is changed, the entire block changes on average by 45% after encryption [4]. This property increases encryption strength, due to the fact that attacks, based on small changes in the source text, stop working.

Block chaining is provided by changing the Huffman table for all following blocks, depending on the results of the encrypted block. So, changing any bit in the block, changes the result of encryption not only of this block, but of all the following. Depending on the Huffman table, changes in subsequent blocks are in range from 40% to 60% and on average provide good block chaining.

C. Cremona transformations

In the existing version, the Cremona transformation is used to increase the entropy of the Generator seeds, which allows use the Generator based on the current time not so often. The key is sufficient to create a whole set of generators, based on these transformations, which can be switched during operation, ensuring the Generator operation unpredictability.

Another option is to generate fake characters at once for a set of blocks. This way will allow to access to the generator less often, which will significantly increase the speed of encryption. Moreover, the fact of working in the fields allows not to worry about a great increase in the transformations degrees, which will provide an acceptable generation rate.

V. PERFORMANCE

The sizes of the read character and block are 16 bits and 8 characters respectively. The key is 256 bits. Modules are: 5, 7, 11, 17, 19. Processor: Intel Core i7-4720HQ 2.6 GHz. The test results for files of different sizes are presented below:

TABLE I.
TEST RESULTS

Algorithm	**File Size (Bytes)**	**Encrypt time (s)**	**Decrypt time (s)**	**Encrypt file size (Bytes)**
Our algorithm	1 048 576 (1 MB)	0.24 – 0.27	0.23 – 0.25	1 726 399 – 1 727 072
AES-256	1 048 576 (1 MB)	0.035	0.068	1 048 576
Our algorithm	104 857 600 (100 MB)	25.12 – 25.96	23.87 – 24.42	172 611 865 – 172 622 351
AES-256	104 857 600 (100 MB)	2.63	5.96	104 857 600

It can be seen that our algorithm loses in speed. This is due to the fact, that the algorithm is probabilistic and, in fact, a greater amount of data is encrypted, than is fed to the input. Also, the current version of the algorithm is just an early prototype, and we are still working on optimization. Nevertheless, even now, because of the great flexibility of the settings, you can achieve a significant speed increase.

VI. CONCLUSION

In this article, a modification of a fully probabilistic cipher based on the theory of information compression and Cremona transformation was presented. This modification can be useful in various areas, since it has a modular structure. Probabilistic encryption guarantees high cryptographic strength for any application. In future research, we hope to optimize this algorithm to make it lightweight, and explore the application of Cremona transformations more. We hope that in the near future there will be an increased interest in probabilistic encryption, so that there is an opportunity to actively develop this direction and compare this algorithm with analogues.

ACKNOWLEDGMENT

We greatly thank our scientific director Sergey Krendelev for support and comments that improved the paper. We would also like to show our gratitude to the team of key exchange, under the direction of Sergey Krendelev, for sharing their ideas with us during the course of this research.

REFERENCES

[1] S Krendelev, N Zbitnev, D Shishlyannikov and D Gridin, "Block cipher based on modular arithmetic and methods of information compression" IOP Conf. Series: Journal of Physics: Conf. Series 913 (2017) 012009 https://doi.org/10.1088/1742-6596/913/1/012009
[2] Vinogradov I M "Elements of Number Theory", 5th ed Kravetz S, Dover, 1954
[3] Nelson M 1995 "The Data Compression Book", 2nd Edition IDG Books Worldwide Inc
[4] Schneier B "Applied Cryptography Second Edition", John Wiley & Sons Inc, 1996
[5] S. Cantat "The Cremona group in two variables", Proceedings of the sixth European Congress of Math., pp. 211–225, Europ. Math. Soc., 2013
[6] S. Cantat "The Cremona Groups", to appear in Proceedings of 2015 Summer Institute on Algebraic Geometry, AMS Proceedings of Symposia in Pure Mathematics