Block cipher based on modular arithmetic and methods of information compression

S Krendelev, N Zbitnev, D Shishlyannikov and D Gridin
Novosibirsk State University, JetBrains research

e-mail: s.f.krendelev@gmail.com, nikita.zbitnev.a@gmail.com, dmitry.shishlyannikov.a@gmail.com, dmitry.gridin.v@gmail.com

Abstract. The article focuses on the description of a new block cipher. Due to the heightened interest in BigData the described cipher is used to encrypt big volumes of data in cloud storage services. The main advantages of the given cipher are the ease of implementation and the possibility of probabilistic encryption. This means that the text encryption will be different when the key is the same and the data is the same. So, the strength of the encryption is improved. Additionally, the ciphered message size can be hardly predicted.

1. Introduction
Standard block ciphers are based on bit data representation and use standard methods of encryption. They are permutation, substitution and some characteristics of Galois fields over the field of characteristic 2. The following methods:
1) Modular methods
2) Fully homomorphic encryption methods
3) Data compression methods
4) p-adic numbers methods to build a non-periodic PRNG. This generator is not used in this work, because it has not yet been possible to increase the speed of the generator.

Let us turn to the description of the algorithm.

2. Algorithm description
It is necessary to encrypt a block of text \( B = q_1, q_2, ..., q_n \), which has \( n \) symbols of \( b \)-bit size each with secret key.

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

\[ l = 2 \times b \times n \] (1)

The key itself is presented as a sequence of integers: \( \beta_1, \beta_2, ..., \beta_k, \beta_{k+1}, ..., \beta_l \), where \( k = \frac{1}{2} \)

To extract all the necessary data from the key we would use an additive generator [1] (hereinafter the Generator) represented as a recurrence relation of the following type:

\[ X_{k+l} = (\beta_1 * X_{k+l-1} + \beta_2 * X_{k+l-2} + ... + \beta_k * X_l) \mod 251, \] (2)
where $X_k, X_{k-1}, ..., X_1 = \beta_l, \beta_{l-1}, ..., \beta_{k+1}$.

2.2. Preparation Phase

2.2.1. Module Choice. Let us choose a set of prime integers $M_1, M_2, ..., M_m$ such as:

$$M = M_1 \cdot M_2 \cdot ... \cdot M_m \in [2^b, 2^{b+1}] \quad (3)$$

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

2.2.2. Generation of Permutations. To provide nonlinear encryption it is necessary to obtain permutations for arithmetic operations for each module from the secret key [3]. To the module mod a permutation is a set of prime integers of the type:

$$(a_0, a_1, a_2, ..., a_{mod-1}), \text{where } a_i \in [1, mod - 1] \text{ and } a_i \neq a_j, i \neq j \quad (4)$$

The set is obtained by induction as:

- $a_0 = 0$;
- $a_{k+1} = a_i$, where $i$ is a number obtained by Generator and taken remainder of $k + 1$;
- $a_k = k + 1$.

A permutation changes addition and multiplication tables for the given module as follows. Let us take addition as an example. In a standard addition table

$$x + y = r \quad (5)$$

And in the table we managed to generate

$$x + y = i, \text{where } a_i = r \quad (6)$$

Multiplying is done analogically.

2.2.3. Generation of Matrices. For each module invertible matrices $A_1, A_2, ..., A_m$ of $n \times n$ size are generated. To do this we use the following recurrence relation (within the framework of generated addition and multiplication tables):

$$z_{n+k} = \alpha_1 * z_{n+k-1} + \alpha_2 * z_{n+k-2} + \cdots + \alpha_n * z_k \quad (7)$$

where $\alpha_i \in [1, mod - 1]$ integers are obtained by the Generator, and $z_n, z_{n-1}, ..., z_1 = (1, 0, 0, ..., 0, 1)$.

So, let us generate using this algorithm $k = 1 + \Sigma_{i=1}^{n} \alpha_i$ vectors. The combination of the latter $n$ vectors is a desired matrix. The obtained matrix is invertible as a matrix obtained from basic data is invertible (an identity matrix), while the following matrices are obtained by linear combination.

2.2.4. Generation of Huffman Table. At a stage of ciphering Huffman tables are used to improve the strength of the algorithm cryptography. This choice may be justified by the fact that the combination of Huffman coding [4] 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 [5]. This helps to resist entropic methods of cracking and known-plaintext attacks.

Building a Huffman table and using it is as follows:

Let us build a frequency table for every symbol. As in our case all ciphered symbols will not exceed the maximum absolute value, it will be enough to build a frequency table for digits from 0
to the maximum module value. It is impossible to count real frequency in the text, because we work with only block of this text so, the frequency of every symbol will be generate by the Generator.

We generate a Huffman tree based on this table:
1) The symbols of the input alphabet make up a list of free nodes. Every node contains the weight represented in the table.
2) Two free nodes of the least weight in the tree are chosen.
3) The weight of the generated parent node is set to the sum of the weight of the children.
4) The parent is added to the queue and two nodes as children are removed from it.
5) Bit ‘1’ represents the left branch growing out from the parent and bit ‘0’ – the right one. Bit marking of the branches growing out from the root does not depend on the weights of ‘descendants’.
6) This process is repeated until only one node remains, which is the root of the Huffman tree.

To find out the code for every symbol included into the message we must make our way from the root of the tree to the node representing the symbol. We collect bits while moving from branch to branch. The sequence of bits thus obtained is the code of the given symbol.

2.2.5. 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.

It is evident that if the size of the symbol is \(b\)-bit, the encrypted symbols cannot have a larger size.

Also, according to the Chinese remainder theory \([2]\), from the remainders of the division of an integer we can collect a number which is smaller than the multiplication of all modules \(M\). Then, if a number is added into a block would satisfy following condition

\[
\lambda: 2^b < \lambda < M
\]

it will be clear that the number is not a symbol of the source file when the block is deciphered and collected.

To implement it, we will produce a new Generator. The starting data for the new Generator are taken from system time as follows: time and date are represented at the moment of ciphering as a format string – \(Www\ Mmm\ dd\ hh:mm:ss\ yyyy\), where \(Www\) is an abbreviated notation for a day of the week, \(Mmm\) stands for a month, \(dd\) is a date, \(hh:mm:ss\) is hours, minutes and seconds respectively. Obtaining a numeric representation for each symbol of the string, we have a set of \(24\) integers. Now these integers are used to initialize the Generator with the above-described algorithm.

Before block reading, we will obtain a number \(\lambda\), from the new Generator and find the number \(\lambda_1 = \lambda \mod n\). If \(\lambda_1 \neq 0\), then \(\lambda_1\) - is a position into which a fake symbol will be put into the red block. After that read \(n - 1\) symbol from the text and filled the block with the fake symbol \(\lambda\) in position \(\lambda_1\).

If \(\lambda_1 = 0\), then the fake symbol is not added, \(n\) symbols are read and the block is filled in the usual way.

3. Ciphering

We will choose modules: \(M_1, \ldots, M_m\). Secret key is used to generate permutations, matrices \(A_1, A_2, \ldots, A_m\) for each module and a Huffman table. Read the block \(B\) of size \(n \times b\) and ciphered as follows:
1) The block \(B = q_1, q_2, \ldots, q_n\) is represented in the form of a vector

\[
B = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{pmatrix}
\]

The coordinates of the vector are remainders of \(M_1, \ldots, M_m\). We would have \(m\) vectors:
\[ v_1 = \begin{pmatrix} v_{11}^1 \\ v_{12}^1 \\ \vdots \\ v_{1n}^1 \end{pmatrix}, v_2 = \begin{pmatrix} v_{21}^2 \\ v_{22}^2 \\ \vdots \\ v_{2n}^2 \end{pmatrix}, \ldots, v_m = \begin{pmatrix} v_{11}^m \\ v_{12}^m \\ \vdots \\ v_{1n}^m \end{pmatrix}, \text{where } v_{ij} = q_{j \mod M_i} \]  

(10)

2) Matrices are multiplied by the relevant vectors (in every module), all operation we do by tables of multiplication and addition generated previously.

\[ w_i = A_i \cdot v_i, \text{where } i = 1..m \]  

(11)

3) The resulting vectors are recorded a special order in accordance with the Huffman table. The order is defined by the sequence of numbers \( P = \{p_1, p_2, \ldots, p_m\} \), 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: \( v_{p_1}, v_{p_2}, \ldots, v_{p_m} \).

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. To do this we get integers \( c \) and \( d \) from the Generator. Then the decision is made which operation to perform by calculating the remainders of the division \( c \) by 3:

- If \( c \mod 3 = 0 \), then the we use permutation of columns \( c \mod n \) and \( d \mod n \) in matrices \( A_1, A_2, \ldots, A_m \).
- If \( c \mod 3 = 1 \), then the we use permutation of strings \( c \mod n \) and \( d \mod n \) in matrices \( A_1, A_2, \ldots, A_m \).
- If \( c \mod 3 = 2 \), then the we use transposition of matrices \( A_1, A_2, \ldots, A_m \).

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

The Huffman table is changed similarly. The symbols in the table remain their positions and their relevant codes are circularly shifted by the number obtained from the Generator.

4. Deciphering

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

1. Read \( m \) encrypted vectors \( v'_1, \ldots, v'_m \) and every symbol is deciphered in accordance with a Huffman table.

2. An invertible matrix is multiplied by the relevant vector, all operation we do by tables of multiplication and addition generated previously:

\[ A_i^{-1} \times v'_i, i = 1, \ldots, m, \]

Result of this operation is deciphered vectors \( v_1, \ldots, v_m \).

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 \( 2^6 \), 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.

5. Cryptanalysis

5.1. PRNG

A simplified recurrent generator is used in our algorithm. It is obviously is not a good one, but it perfectly answers our needs. This Generator has been described in a number of articles [1]. Their main finding to which we will refer is the PRNG period. Parameters:

1) Module – 251, a prime number.
2) Recurrence relation length – 8. So, the estimated period is $251^8 \approx 10^{18}$.

3) At the pre-ciphering phase the Generator is called $8 + \sum_{i=1}^{m} M_i + \max_{i \in [1,m]} M_i$ times.

4) At the ciphering phase the Generator is called four more times.

Estimation: The Generator is called about $10^7$ times for a file of 1 GB in size, and the Generator period is $10^{18}$. It enables us to use the Generator for our algorithm.

5.2. Huffman and Influence of Fake Symbols

Fake symbols added to the block before ciphering do not influence occurrence of irrepresentable integers in encrypted vectors (an integer more than $2^{16}$, when the size of symbol is 16 bits). The probability of their occurrence is always equal and depends only on the choice of a module set:

$$\prod_{i=1}^{m} M_i^{-2^b} \prod_{i=1}^{m} M_i \approx 45\%$$

(12)

for modules of: 5, 7, 11, 17, 19 and size of symbols 16 bits.

5.3. Avalanche effect

Any bit in any block is changed, the whole block changes by 40 percent on average (based on available statistics). This is very close to the definition of Avalanche effect [5]. When a bit is changed, its relevant coordinate in vectors in modules changes. As all coordinates take part in multiplication of matrix by vector, any change of a coordinate leads to a complete change of a resulting vector.

5.4. Modified Brute Force

The classical Hill cipher is vulnerable to a known-plaintext attack. To force the attacker to do more work to break our algorithm we use the permutations of addition and multiplication tables. To restore $i$-th matrix after the permutations we need $(M_i - 1) \times n$ texts, where $M_i$ – $i$-th module. Then a multiplication table must be built up and the permutations must be restored. This gives rise to two cases:

1) If all the elements occur in the matrix, the table is restored by using

$$(M_i^2 + 1) \times 2^{-1} - 2 \times (M_i - 1)$$

(13)
texts.

2) If $k$ elements are absent, an extra text $(k - 1) \times M_i$ is needed.

Using a Huffman table helps hide vectors. First, it is necessary to restore the Huffman table or the ciphering tree to recognize vectors by the encrypted text. So, if the tree-depth is $l$ there are $C_{2^l}^{M_{\max}}$ ways to allocate $M_{\max}$ elements, where $M_{\max} –$ is a maximum module. After the table/tree is restored symbols and ciphers must be compared. The total number of variants is $M_{\max}!$.

The resulting formula to crack our algorithm is:

$$C_{2^l}^{M_{\max}} \times M_{\max}! \times \prod_{i=1}^{m} \left( (M_i - 1) \times n \times \left( \frac{M_i^2 + 1}{2} - 2 \times (M_i - 1) \right) \right)$$

(14)

The strength of the encryption for a set of modules (5, 7, 11, 17, 19) and a 6-bit Huffman code as the maximum size may be evaluated as $2^{181}$.

6. Performance

A series of tests helped us find the optimum sizes of symbols and blocks. They are 16 bits for a symbol and 4 symbols for a block. Let us examine the performance of our algorithm:
6.1. Tests

Modules: 5, 7, 11, 17, 19.

| File size (MB) | Number of blocks | Number of fake symbols | Encrypt time (s) | Decrypt time (s) | Encrypt file size (MB) |
|----------------|------------------|------------------------|-----------------|-----------------|-----------------------|
| 1              | 131 072          | 138 210                | 0.21            | 0.28            | 1.7                   |
| 32             | 4 194 304        | 4 233 113              | 7.02            | 8.72            | 54.05                 |
| 1024           | 134 217 728      | 141 451 136            | 217             | 268             | 1 744                 |

The average speed of ciphering is 5 566 634 blocks per second. It takes 790 operations to cipher one block. Thus, the performance is $4.4 \times 10^9$ operations per second. Taking into account program parallelism, the performance can be increased $m$ times, where $m$ is a number of modules.

7. Conclusion

In this paper, we create new fully probabilistic encryption algorithm for working with Big Data. Probabilistic encryption guarantees high cryptographic strength. Further research will be related to the development of random number generators not having a period and using MTF (Move-To-Front) technology from the theory of data compression [4], to generate permutations.

References

[1] Dessai V, Ravindra P and Dandina R 2012 Using Layer Recurrent Neural Network to Generate Pseudo Random Number Sequences *IJCSI International Journal of Computer Science Issues* **9** p 324–334

[2] Vinogradov I M 1954 *Elements of Number Theory* (5th ed Kravetz S, Dover)

[3] Farmanbar M and Chefranov A 2012 Investigation of Hill Cipher Modifications Based on Permutation and Iteration *International Journal of Computer Science and Information Security* **10** (9) p 18–24

[4] Nelson M 1995 *The Data Compression Book* (2nd Edition IDG Books Worldwide Inc)

[5] Schneier B 1996 *Applied Cryptography Second Edition* (John Wiley & Sons Inc)