An extension of the class of Boolean functions used in symmetric cipher algorithms

Svetlana Korabel'shchikova
Northern (Arctic) Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
s.korabelsschikova@narfu.ru

***Abstract*—One of the standard cryptographic transformations is the addition modulo two of an open binary text with a key binary sequence. In this paper, we have obtained a description of all Boolean functions from n arguments that are suitable for use in cryptographic transformations instead of the modulo-two addition function (we will call them component-by-component Boolean functions). An example of their use in the cryptographic conversion algorithm GOST R 34.12-2015 is also given. The article proposes an algorithm for generating component-by-component Boolean functions from n variables for different values of n and k, where k is the number of the variable whose value the function returns. For the case n=3 and k=1 or k=2, all Boolean functions that replace the addition function modulo 2 are represented. The paper proposes an encryption method based on component-by-component Boolean functions and a pseudo-random sequence generator of elements from the GF(2^{n-1}) field. Using Boolean functions that return the value of one of the arguments when repeated, expands the variety of intermediate variants of round transformations, gives a new variable encryption method, ultimately significantly increasing the cryptographic strength of the cipher.**

Keywords—boolean functions, encryption, symmetric cipher, GOST R 34.12-2015

I. INTRODUCTION

To date, many works on cryptography have been devoted to Boolean functions, the study of certain cryptographic properties of Boolean functions, as well as the possibilities of their use in order to protect information. These issues are discussed both in scientific articles and in a number of textbooks for Universities, for example, [1, 2]. For the most complete overview of the cryptographic properties of Boolean functions and available results see [3]. This article is a continuation of work [4], in which the authors proposed Boolean functions of three arguments that replace the addition operation modulo *2*. In [5], the author has previously considered the issues of information security using linear encoding.

One of the standard cryptographic transformations used in various symmetric encryption algorithms is bitwise addition modulo two of a plaintext represented in binary form with a key binary sequence. For example, GOST R 34.12-2015 [6] is a symmetric cipher in which the beginning of the transformation sequence is the transformation *X[k]*:

$$X[k](a) = k \oplus a, \quad (1)$$

where k is the round key, a is the plaintext, k, $a \in V_{128}$.

The decryption method consists in the repeated addition modulo 2 of the ciphertext with the same key k:

$$D\,X[k](a) = (k \oplus a) \oplus k = a. \quad (2)$$

We use the same Boolean function to encrypt and decrypt information:

$$F(x, y)=x \oplus y. \quad (3)$$

This function has the following property:

$$F(F(x, y), y) = x. \quad (4)$$

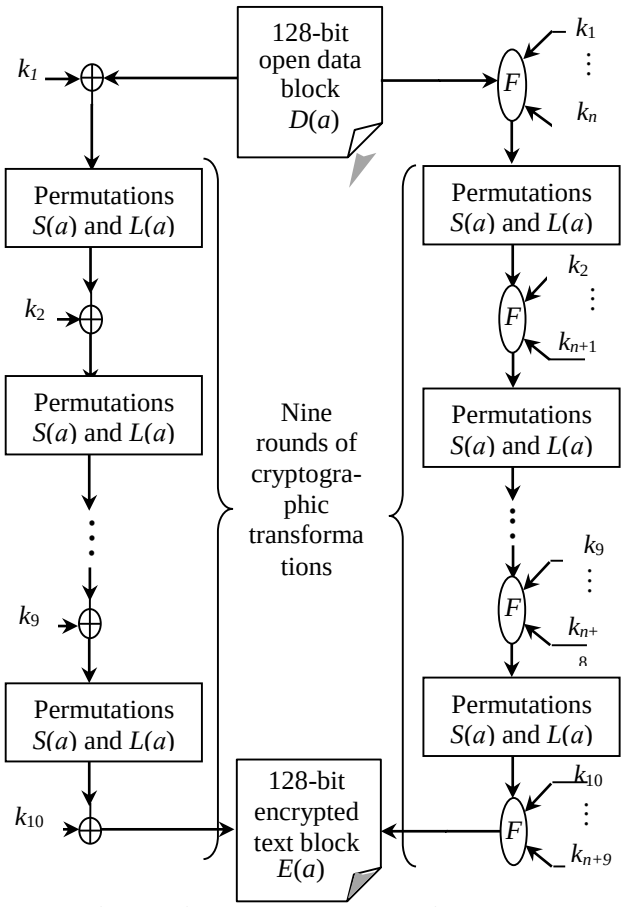

Fig. 1. Modification of the encryption algorithm from GOST R 34.12-2015 using component-wise functions.

Earlier in [4], we proposed 10 Boolean functions of three variables to replace addition modulo 2. We introduced the term component-wise functions for them. In general case, a Boolean function of n variables, which returns the value of the first argument, must satisfy the condition:

$$F(F(x_1, x_2, \dots, x_n), x_2, \dots, x_n) = x_1. \quad (5)$$

The figure 1 shows the possible location of the component functions F_j, which depend on $n+1$ arguments, in the process of encrypting a 128-bit block of information. We presented two algorithms: an algorithm from GOST R 34.12-2015 standard (left) and a modification proposed by the authors (right).

Modern studies of the GOST R 34.12-2015 cipher are presented in [7- 9] and others. They are mainly devoted to the study of the cryptographic strength of this encryption standard, as well as various options for its software implementation. This article describes all Boolean functions of n arguments that are suitable for use in symmetric cryptographic transformations instead of the modulo two addition function. We proposed an algorithm for their generation, an encryption method based on component-wise

Boolean functions and a generator of a pseudo-random sequence of elements of the field $GF(2^{n-1})$.

II. The generation algorithm and properties of component-wise functions

Condition (5) means that the system of equations:

$$\begin{cases} F(F(0, x_2, \dots, x_n), x_2, \dots, x_n) = 0 \\ F(F(1, x_2, \dots, x_n), x_2, \dots, x_n) = 1' \end{cases} \quad (6)$$

whence it follows that $F(0, x_2, \dots, x_n) = 0$ if and only if $F(1, x_2, \dots, x_n)$=1 and vice versa, $F(0, x_2, \dots, x_n)$=1 if and only if $F(1, x_2, \dots, x_n)$=0. We will represent the Boolean function F by the vector of values $\tilde{F}$=(α_0,α_1,…, α_{2^n-1}), written in the order of correspondence with the ordered sets of variable values from zeros to ones. Then the conditions obtained mean that if the vector of values of the component-wise Boolean function is divided into two parts, then the second part will be inverse to the first. Therefore, the following statement is true.

Theorem 1. The number of Boolean functions of n variables satisfying condition (5) is $2^{2^{n-1}}$. A Boolean function satisfies condition (5) if and only if the second half of its value vector is inverse to the first.

Let us prove the first statement. We can define a Boolean function of n variables by its vector of values of length 2^n. Since the second half of the value vector completely depends on the first (inverse to the first), you can set it by specifying the first half of the value vector in an arbitrary way. It has a length of 2^{n-1}, and the number of binary vectors of this length is $2^{2^{n-1}}$.

Let us prove the second statement of the theorem. The values of $F(0, x_2, \dots, x_n)$ are in the first half of the vector $\tilde{F}$, and in the same order in the second half of the vector $\tilde{F}$ are the values of $F(1, x_2, \dots, x_n)$. From the conditions obtained above, it follows that $F(0, x_2, \dots, x_n) = \overline{F(1, x_2, \dots, x_n)}$, that is, the second half of the vector $\tilde{F}$ is inverse to the first.

The theorem is proved.

Example 1. For n=2 we have $2^{2^{2-1}}$=4 Boolean functions that return the first argument. We list their value vectors. The first half of the vector of values is set arbitrarily; the second is completed inversely with the first: 0011, 0110, 1001, 1100.

We got the following 4 functions: $F(x, y) = x$, $F(x, y) = x+y$, $F(x, y) = x \leftrightarrow y$ and $F(x, y) = \bar{x}$.

Example 2. For n=3 we have $2^{2^{3-1}}$ =16 Boolean functions that return the first argument. They have value vectors: 00001111, 00011110, 00101101, 00111100, 01001011, 01011010, 01101001, 01111000, 10000111, 10010110, 10100101, 10110100, 11000011, 11010010, 11100001 and 11110000.

Taking into account the requirements for cryptographic functions, we come to conclusions from examples 1 and 2. Note that the first and the last functions are the negation of each other. The second and penultimate functions are related in a similar way, and so on. Thus, if we are going to use several component-wise functions in the encryption algorithm, we will select them only one from each pair.

It is also obvious that some of the functions obtained contain dummy variables, which is unacceptable for cryptographic functions. Applying the algorithm for determining the fictitiousness of the variables x_2, ..., x_n to the selected functions, we obtain all Boolean functions that satisfy condition (5) and substantially depend on all variables.

Definition 1. A component-wise (n, k) function, where $1 \le k \le n$, is a Boolean function $F(x_1, x_2, \dots, x_n)$, that does not contain fictitious variables and returns the k-th argument when it is reused, i.e., satisfying the condition:

$$F(x_1, x_2, \dots, x_{k-1}, F(x_1, x_2, \dots, x_n), x_{k+1}, \dots, x_n) = x_k \quad (7)$$

We formulate an algorithm for generating all component-wise functions $F(x_1, x_2, \dots, x_n)$, that substantially depend on n variables and return the variable x_k when reused. The input of the algorithm: n is the number of variables, k is the number of the variable whose value the function returns.

The generation algorithm

Step 1. Enter n, k.

Step 2. Calculate 2^{n-1} and generate all possible binary vectors of length 2^{n-1}.

Step 3. (Checking for fictitiousness of all variables). For verification, we use the algorithm from [10]. We delete all vectors that did not pass verification.

Step 4. (Adding inverted parts) Run for all vectors remaining after step 3.

Divide the vector into 2^{k-1} parts and add the inverse one after each part.

Include all received vectors in the response.

In the proposed algorithm, we check for fictitiousness of variables only half of the value vector. If the check was successful, then all the variables will be significant for the full value vector generated in step 4.

Example 3. Here is an example of how the algorithm works for n=3, k=2.

2^{3-1}=4, so we generate all binary vectors of length 4:

0000, 0001, 0010, 0011, ..., 1111.

After step 3, only 10 vectors that have passed verification will remain out of the 16 vectors:

0001,0010,0100,0110, 0111, 1000, 1001, 1011, 1101, 1110.

Performing step 4, divide each vector into 2 parts, and add an inversion to each part. We get the following result:

TABLE I. *(3,2)* COMPONENT-WISE FUNCTIONS

	(3,2) component-wise functions		*(3,2)* component-wise functions
0001	00**11**01**10**	1000	10**01**00**11**
0010	00**11**10**01**	1001	10**01**01**10**
0100	01**10**00**11**	1011	10**01**11**00**
0110	01**10**10**01**	1101	11**00**01**10**
0111	01**10**11**00**	1110	11**00**10**01**

Let us show that we actually got functions that return the second component. Take, for example, $F_1(x_1, x_2, x_3)$, $\tilde{F}_1$=(00110110). We show that for arbitrary values x_1 and x_3 and for $x_2=a$, the function F_1 returns the value a.

If we take $n=3$ and $k=1$ in the algorithm, the first steps of the algorithm will be the same as in example 3. In step 4, we do not split the remaining 10 vectors, since $2^{k-1}=2^0=1$. Add the inverse part to them and get the vectors of the values of (*3,1*) functions from example 2, with the exception of six functions containing fictional variables.

(x_1, x_3)	$F_1(x_1, 0, x_2)$	$F_1(x_1, F_1(x_1, 0, x_3), x_3)$
00	0	0
01	0	0
10	0	0
11	1	0

(x_1, x_3)	$F_1(x_1, 1, x_2)$	$F_1(x_1, F_1(x_1, 1, x_3), x_3)$
00	1	1
01	1	1
10	1	1
11	0	1

All component-wise (*n, k*) functions are balanced, that is, they take the values 0 and 1 equally often. However, they have a different probability of replacing or saving the plaintext character. So, the function from example 3 with the value vector $\tilde{F}_1$=(00110110) changes the value of the plaintext a only in 2 cases out of 8, that is, it has a 25% the probability of replacing the plaintext.

The best characteristics (50% to 50%) of transition probabilities are those Boolean functions that correspond to the balanced vectors that remain after step 3 of the above algorithm.

Example 4. When *n*=4 of the 256 vectors generated in step 2, only 220 are checked for the absence of fictive variables. Of these, 58 vectors are balanced. At n=4, it is balanced and passes the check for the absence of fictive variables, for example, the vector 11010100. This means that (*4,1*) a component-wise function with the value vector 11010100**00101011** has a 50% to 50% transition probability. The same probability of transitions have (*4,2*) component function with the value vector 1101**0010**0100**1011**, (*4,3*) component function with the value vector 11**00**01**10**01**10**00**11**, (*4,4*) component function with the value vector 1**0**1**0**0**1**1**0**0**1**1**0**0**1**0**1**.

Let *r* be a non-negative number less than *n*. A Boolean function f from *n* variables is called *r*-stable if any of its subfunctions obtained by fixing at most *r* variables are balanced.

Theorem 2. For any component-wise *(n,k)* function $F(x_1, x_2, \dots, x_n)$ the following conditions are equivalent:
$F(x_1, x_2, \dots, x_n)$ has transition probabilities 50%;
$F(x_1, x_2, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n)$ is balanced;
$F(x_1, x_2, \dots, x_n)$ is 1-stable.

We prove 1⇒2. The length of the vector of values of the function $F(x_1, x_2, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n)$ is 2^{n-1}. Let the vector of values of the function $F(x_1, x_2, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n)$ have t units. It is necessary to prove that $t = 2^{n-2}$, i.e., that the number of units is exactly half the length of the vector. Since (6) holds, the vector of values of the function $F(x_1, x_2, \dots, x_{k-1}, 1, x_{k+1}, \dots, x_n)$ has t zeros. Then the total number of substitutions is 2t. By condition, the probability of substitutions is 50%, that is $2t = 2^{n-1}$, whence $t = 2^{n-2}$.
We prove 2⇒3. Note that it is the vector of values of the function $F(x_1, x_2, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n)$ that we obtain at step 3 of the algorithm proposed above. Since, by condition, it is balanced, the inverse vector $F(x_1, x_2, \dots, x_{k-1}, 1, x_{k+1}, \dots, x_n)$ will also be balanced.

Now we consider the function $F(0, x_2, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_n)$. It can be divided into two subfunctions: $F(0, x_2, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n)$ and $F(0, x_2, \dots, x_{k-1}, 1, x_{k+1}, \dots, x_n)$, and the second will be inverse to the first. Therefore, the combined vector $F(0, x_2, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_n)$ has an equal number of zeros and ones. It is proved in a similar way that the vector of values of the function $F(1, x_2, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_n)$ is balanced. For the same reason, the vectors of subfunction values are balanced, which are obtained by fixing one of the variables $x_2, \dots, x_{k-1}, x_{k+1}, \dots, x_n$. Therefore, $F(x_1, x_2, \dots, x_n)$ is 1-stable.

We prove 3⇒1. It follows from the condition that the function $F(x_1, x_2, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n)$ is balanced, that is, takes 2^{n-2} times the value 1 and 2^{n-2} times the value 0. In addition, the function $F(x_1, x_2, \dots, x_{k-1}, 1, x_{k+1}, \dots, x_n)$ is balanced, that is, takes 2^{n-2} times the value 1 and 2^{n-2} times the value 0. Then the number of substitutions is 2^{n-1}, which means the probability of transitions is 50%.
The theorem is proved.

III. Encryption options based on exploded Boolean functions

One of the encryption options using *(3, 1)* component-wise Boolean functions was proposed by us in [4], and is shown schematically in Figure 1. However, to use component-wise *(n, k)* functions for encryption, it is necessary to have or generate n –1 key binary sequence, which is not always convenient. Instead, one pseudo-random sequence of elements of the field *GF* (2^{n-1}) can be generated. To do this, you can use, for example, a PSP generator based on linear registers of shift registers (LFSR).

Let $F(x_1, x_2, \dots, x_n)$ – be a component-wise *(n,1)* function, *a* –be binary plaintext, K be the key SRP of the elements of the field *GF* (2^{n-1}). Then the encryption of the text a is performed elementwise according to the formula:

$$E(a) = F(a, K). \quad (8)$$

To decrypt, we use the same function $F(x_1, x_2, \dots, x_n)$ and the key sequence *K*:

$$D(E(a)) = F(F(a, K), K) = a. \quad (9)$$

The organization of calculations largely depends on the representation of the elements of the field *GF* (2^{n-1}), therefore, we will consider this question in more detail. Operations on elements of a finite field *GF* (2^m) are easily performed when they form an index table, where *m*-dimensional binary vectors are associated with the powers of a primitive element. Such a representation is also convenient when dividing field elements into circular classes, used, for example, to find primitive polynomials or in algorithms for obtaining the number of noise-resistant codes [11].

Consider the algorithm for constructing the index table of the field *GF*(2^m), where $m \leq 30$. Input data: *m* is the degree of expansion, *f(x)* is the primitive polynomial of degree *m* over *GF(2)*. At the end of the algorithm, the program memory contains all 2^m-1 nonzero vectors of length m in a certain order, specifically, in powers of the primitive element α – the root of the polynomial *f(x)*.

To optimize the speed of calculations, we will store each vector of coefficients in a 32-bit integer data type. At the position of the *i*-th bit in the binary notation of the number, the *i*-th coefficient of the vector will be stored. Due to this, the multiplication of the vector by α will be carried out by a bitwise left shift. The search for the remainder of division

by the primitive polynomial $f(x)$ will be expressed through the operation of bitwise addition modulo 2. Thus, the storage of the index table $GF(2^m)$ requires about $4 \cdot 2^m$ bytes of memory.

Note that the calculation of each subsequent coefficient vector is performed sequentially. Thus, lines 0, 1, 2, …, 2^m-3, 2^m-2. are filled. In order to calculate the table using parallel technologies, we break all the rows into k consecutive blocks. To calculate the j-th block, you need to calculate the coefficient vector that is the first in this block. For this, there is no need to find all previous vectors. We use the binary exponentiation algorithm to significantly speed up the calculations.

We tested the work of the program for constructing the index table of the $GF(2^m)$, where m=26, 27, 28, 29 and 30. The calculations were performed on 4 cores on an NArFU cluster with 20 computing nodes, each of which had 2 10-core Intel Xeon processors and 64 GB of RAM. The table contains data on the operating time for various m and for a different number of threads.

TABLE II. EXECUTION TIME OF SERIAL AND PARALLEL ALGORITHMS (IN SECONDS)

Threads \ m	**26**	**27**	**28**	**29**	**30**
Sequential algorithm	0.507	1.003	2.013	4.034	8.290
1 thread	0.535	1.077	2.143	4.283	8.869
2 threads	0.311	0.624	1.247	2.495	5.103
3 threads	0.242	0.473	0.945	1.910	3.877
4 threads	0.205	0.403	0.794	1.597	3.283

From the presented table we can conclude that the program shows an acceleration of about 2,5 times with parallel implementation of 4 threads.

IV. CONCLUSION

Component-wise (n, k) functions extend the modes of standard cryptographic transformations, in particular, GOST R 34.12-2015. Using them instead of the modulo 2 addition operation increases the possibilities of choosing round transformations for symmetric ciphers. But a significant disadvantage of the functions under consideration is a large redundancy. We believe that consideration of K- digit component-wise functions will help overcome this disadvantage. Further research will be aimed at introducing the encryption method using component-wise functions in the hardware-software complex.

REFERENCES

[1] N.N. Tokareva, “Simmetrichnaya kriptografiya,” Novosibirsk: NSU, 2012, 232 p.

[2] S.N. Selezneva, “Multiplicative complexity of some functions of the algebra of logic,” Discrete Mathematics, vol. 26, no. 4, pp. 100-109, 2014.

[3] A.A. Gorodilova, “From cryptanalysis of a cipher to the cryptographic property of a Boolean function,” Applied Discrete Mathematics, vol. 3, no. 33, pp. 4-44, 2016.

[4] I.I. Vasilishin and S.Yu. Korabelshchikova, “Using component-wise function in cryptographical transformation algorithm from Russian national standard GOST R 34.12-2015,” CEUR Workshop Proceedings, vol. 2212, pp. 392-398, 2018.

[5] S.Y. Korabelshchikova, L.V. Zyablitseva, B.F. Melnikov and S.V. Pivneva, “Linear codes and some their applications,” Journal of Physics: Conference Series, 012174, 2018.

[6] GOST R 34.12-2015. Information technology. “Cryptographic information security. Block ciphers,” M .: Standartinform, 2015, 25 p.

[7] E.A. Ishchukova, L.K. Babenko and M.V. Anikeev, “Fast Implementation and Cryptanalysis of GOST R 34.12-2015 Block Ciphers,” 9th International Conference on Security of Information and Networks SIN, Newark, Nj, pp. 104-111, 2016.

[8] T. Isobe, “A single-key attack on the full GOST block cipher,” Journal of Cryptology, vol. 26, pp. 172-189, 2013.

[9] J. Kim, “On the security of the block cipher GOST suitable for the protection in U-business services,” Personal and ubiquitous computing, vol. 17, pp. 1429-1435, 2013.

[10] S.V. Yablonskiy, “Vvedeniye v diskretnuyu matematiku,” M.: Nauka, 2005, 384 p.

[11] B.F. Melnikov and S.Yu. Korabelshchikova, “Algorithms for estimation the number of noise-immune codes of general and special types,” Informatization and communication, vol. 1, pp. 55-60, 2019.