МАТЕМАТИЧЕСКИЕ ВОПРОСЫ КРИПТОГРАФИИ
2021 Т. 12 № 2 С. 21–38

УДК 519.719.2 DOI https://doi.org/10.4213/mvk354

A compact bit-sliced representation of Kuznyechik S-box

O. D. Avraamova[1], D. B. Fomin[2], V. A. Serov[1], A. V. Smirnov[1], V. N. Shokov[1]

[1]*Lomonosov Moscow State University*, *Russia*
[2]*Higher School of Economics*, *Moscow*, *Russia*

Получено 25.XI.2020

Abstract. In this paper we consider a bit-sliced implementation of the non-linear transformation shared by GOST R 34.12-2015 “Kuznyechik” block cipher and GOST R 34.11-2012 “Streebog” hash function. We combine analytical and computer methods to get a 226 Boolean operations representation.

Keywords: block cipher, hash function, S-box, Kuznyechik, Streebog, bit-slice, Boolean functions minimization

Об одном представлении нелинейного преобразования алгоритма «Кузнечик» с помощью логических функций

О. Д. Авраамова[1], Д. Б. Фомин[2], В. А. Серов[1], А. В. Смирнов[1], В. Н. Шоков[1]

[1]*Московский государственный университет имени В. А. Ломоносова*, *Москва*
[2]*НИУ Высшая школа экономики*, *Москва*

Аннотация. Рассматриваются способы реализации нелинейного преобразования блочного алгоритма шифрования с длиной блока 128 бит «Кузнечик» (ГОСТ Р 34.12-2015) и хеш-функции «Стрибог» (ГОСТ Р 34.11-2012). Показана возможность реализации подстановки за 226 логических операций.

Ключевые слова: блочный шифр, хеш-функция, подстановка, Кузнечик, Стрибог, битовая реализация, минимизация представления булевой функции

1. Introduction

A permutation is a bijective mapping of a non-empty finite set onto itself. In applications, permutations of V_n, where V_n is an n-dimensional vector space over Galois field $GF(2)$, are often used. Usually the dimension of the vector space is not large, most popular versions being $n = 4$ or $n = 8$. Permutation may be implemented as a lookup table and this representation is often called an S-box.

S-boxes are widely used in the synthesis of block ciphers and hash-functions. They allow one to combine necessary non-linearity with reasonable implementation complexity of the overall scheme. At the same time, cipher strength against known methods of cryptanalysis depends strongly on certain properties of S-boxes used, and its potential speed and lower resource-consumption — on the effectiveness of software and hardware implementation of these elements.

We use the number of Boolean operations as the measure of complexity of an S-box implementation. In this paper we consider implementations of the non-linear transformation of «Kuznyechik» GOST R 34-12.2015 [1] in the basis of AND, OR, NOT and XOR Boolean functions and try to get a way to make it as compact as possible. The choice of these four functions as the basis is justified by the existence of its effective hardware and software implementation on different platforms. Inside formulas, we'll use short notation: $\cdot$ for AND, $+$ for OR, $\neg$ for NOT and $\oplus$ for XOR.

In the classical problem of Boolean functions minimization the basis is formed by conjunction (AND), disjunction (OR) and negation (NOT). In this form the problem is intimately connected with the simplification of electrical schemas. Karnaugh charts used to complete the problem for electrical schemas are quite usable for manual optimization of Boolean functions with small number of variables. Computer analog of this method was developed by W. Quine and extended by E. McCluskey and is known as Quine-McCluskey algorithm. The distinctive feature of our situation is the existence of additional XOR operation, so finding transformation rules for formulae minimization for the new set of operations is a reasonable problem.

Composition of the paper is the following. In section 1 we consider the decomposition of «Kuznyechik» permutation found by A. Biryukov, L. Perrin, and A. Udovenko in [2], which we call the BPU-decomposition. In section 2 we implement linear and bilinear elements of BPU-decomposition and eliminate branching. In section 3 we consider ways of computer optimization of non-linear elements of the decomposition. In concluding section we compare the overall complexity of our method with other existing solutions.

2. BPU-decomposition

In «Kuznyechik» algorithm, by GOST R 34-12.2015, a non-linear transformation π of vector space V_8 is used. If we try to minimize it directly, using, for example, Espresso algorithm (in Logic Friday [3] realization), we get 3492 Boolean operations — quite a lot. So some other approach is needed.

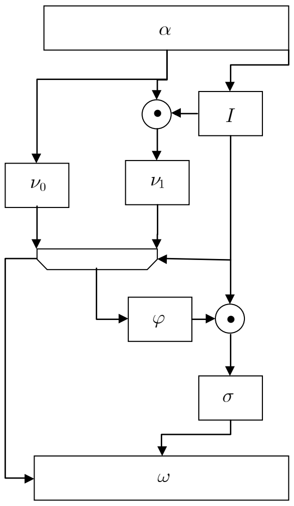

Fig. 1: The structure of BPU-decomposition taken from [2]

The authors of [2] suggested to represent π as a composition of non-linear transformations ν_0, ν_1, I, σ, φ of V_4, linear transformations α and ω of V_8 and multiplication $\circledcirc$ in Galois field $GF(2^4, \circledcirc, \oplus) = GF(2)[x]/(f(x))$ with irreducible polynomial $f(x) = x^4 \oplus x^3 \oplus 1$, where $\oplus$ is the addition and $\circledcirc$ is the multiplication in the finite field.

Consider the action of π on the set of pairs $(l, r), l, r \in GF(2^4)$. In [2] the following algorithm to compute $\pi(l \parallel r)$ was used (see fig. 1):

1) $(l \parallel r) := \alpha(l \parallel r)$,

2) if $r = 0$, then $l := \nu_0(l)$, else $l := \nu_1(l \circledcirc I(r))$,

3) $r := \sigma(r \circledcirc \varphi(l))$,

4) $(l \parallel r) := \omega(l \parallel r)$.

The value of $(l \parallel r)$ after step 4 equals to $\pi(l \parallel r)$. The non-linear transformations (not all of them are bijective) ν_0, ν_1, I, σ, φ are given in the following table (in hexadecimal representation):

I	0, 1, c, 8, 6, f, 4, e, 3, d, b, a, 2, 9, 7, 5
ν_0	2, 5, 3, b, 6, 9, e, a, 0, 4, f, 1, 8, d, c, 7
ν_1	7, 6, c, 9, 0, f, 8, 1, 4, 5, b, e, d, 2, 3, a
φ	b, 2, b, 8, c, 4, 1, c, 6, 3, 5, 8, e, 3, 6, b
σ	c, d, 0, 4, 8, b, a, e, 3, 9, 5, 2, f, 1, 6, 7

3. Implementation of linear and bilinear elements of decomposition and branching elimination

3.1. Bit-sliced implementation of multiplication in the finite field

One of important elements of the decomposition is the multiplication in the finite field $GF(2^4)$. It is used twice during the algorithm. Let's find the representation of this operation by Boolean functions. To do it, we consider $GF(2^4)$ as a four-dimensional algebra over $GF(2)$. In this case the components of binary representation of elements of $GF(2^4)$ will be simply the coordinates of these vectors in the standard basis $\{\mathbf{e}_1 = (1000) = \mathbf{8}, \mathbf{e}_2 = (0100) = \mathbf{4}, \mathbf{e}_3 = (0010) = \mathbf{2}, \mathbf{e}_4 = (0001) = \mathbf{1}\}$. (In this section elements of $GF(2^4)$ will be typesetted in bold and coordinate indexes will be written above).

By the associative, commutative and distributive laws of $GF(2^4)$, operations $\circledcirc$ and $\oplus$ are commuting, and the distributive law holds. So it is sufficient to know pair-wise products of basic vectors, and the product of any two arbitrary vectors may be computed by linearity:

$$\mathbf{z} = (\mathbf{x} \circledcirc \mathbf{y}) = \left(\sum_{i=1}^{4} x^i \mathbf{e}_i\right) \circledcirc \left(\sum_{j=1}^{4} y^j \mathbf{e}_j\right) = \sum_{i,j} x^i \cdot y^j (\mathbf{e}_i \circledcirc \mathbf{e}_j)$$

in vector form and

$$\mathbf{z}^k = (\mathbf{x} \circledcirc \mathbf{y})^k = \left(\left(\sum_{i=1}^{4} x^i \mathbf{e}_i\right) \circledcirc \left(\sum_{j=1}^{4} y^j \mathbf{e}_j\right)\right)^k = \left(\sum_{i,j} x^i y^j (\mathbf{e}_i \circledcirc \mathbf{e}_j)\right)^k$$
$$= \sum_{i,j} x^i \cdot y^j (\mathbf{e}_i \circledcirc \mathbf{e}_j)^k, \quad k = 1, \ldots, 4,$$

in coordinate form.

Let's make a table of pair-wise products of basic vectors:

	1000	0100	0010	0001
1000	1111	1011	1001	1000
0100	1011	1001	1000	0100
0010	1001	1000	0100	0010
0001	1000	0100	0010	0001

Separating coordinates, we have

C^1				C^2				C^3				C^4			
1	1	1	1	1	0	0	0	1	1	0	0	1	1	1	0
1	1	1	0	0	0	0	1	1	0	0	0	1	1	0	0
1	1	0	0	0	0	1	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1

and for every coordinate z^k $(k = 1, \ldots, 4)$ we have

$$z^k = (x^1, x^2, x^3, x^4) \begin{pmatrix} c^k_{11} & \cdots & c^k_{14} \\ \vdots & \ddots & \vdots \\ c^k_{41} & \cdots & c^k_{44} \end{pmatrix} \begin{pmatrix} y^1 \\ y^2 \\ y^3 \\ y^4 \end{pmatrix}.$$

Now (returning to lower coordinate indexes) we are able to write down formulae for every coordinate:

$$\begin{aligned}
z_1 &= (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \cdot y_1 \oplus (x_1 \oplus x_2 \oplus x_3) \cdot y_2 \oplus (x_1 \oplus x_2) \cdot y_3 \oplus x_1 \cdot y_4, \\
z_2 &= x_1 \cdot y_1 \oplus x_4 \cdot y_2 \oplus x_3 \cdot y_3 \oplus x_2 \cdot y_4, \\
z_3 &= (x_1 \oplus x_2) \cdot y_1 \oplus x_1 \cdot y_2 \oplus x_4 \cdot y_3 \oplus x_3 \cdot y_4, \\
z_4 &= (x_1 \oplus x_2 \oplus x_3) \cdot y_1 \oplus (x_1 \oplus x_2) \cdot y_2 \oplus x_1 \cdot y_3 \oplus x_4 \cdot y_4.
\end{aligned}$$

So we need 13 operations to compute z_1, 7 for z_2, 8 for z_3, and 10 for z_4. Multiplication in our realization of $GF(2^4)$ requires 38 Boolean operations if we do not use auxiliary variables to store intermediate results. If we use intermediate variables (denote them by P with indices), the number of operations may be reduced further. For example, the expression $(x_1 \oplus x_2)$

appears more than once:

$$\begin{aligned}
z_1 &= (P_2 \oplus x_4) \cdot y_1 \oplus P_2 \cdot y_2 \oplus P_1 \cdot y_3 \oplus x_1 \cdot y_4,\\
z_2 &= x_1 \cdot y_1 \oplus x_4 \cdot y_2 \oplus x_3 \cdot y_3 \oplus x_2 \cdot y_4,\\
z_3 &= P_1 \cdot y_1 \oplus x_1 \cdot y_2 \oplus x_4 \cdot y_3 \oplus x_3 \cdot y_4,\\
z_4 &= P_2 \cdot y_1 \oplus P_1 \cdot y_2 \oplus x_1 \cdot y_3 \oplus x_4 \cdot y_4,\\
P_1 &= x_1 \oplus x_2,\\
P_2 &= x_1 \oplus x_2 \oplus x_3 = P_1 \oplus x_3.
\end{aligned}$$

The latter realization has complexity 31.

3.2. Implementation of linear mappings

Let's estimate the complexity of linear transformations α and ω in terms of Boolean functions. Matrix representations of α and ω are the following:

$$\alpha = \begin{pmatrix} 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&1\\ 0&1&0&0&0&0&1&1\\ 1&1&1&0&1&1&1&1\\ 1&0&0&0&1&0&1&0\\ 0&1&0&0&0&1&0&0\\ 0&0&0&1&1&0&1&0\\ 0&0&1&0&0&0&0&0 \end{pmatrix}, \quad \omega = \begin{pmatrix} 0&0&0&0&1&0&1&0\\ 0&0&0&0&0&1&0&0\\ 0&0&1&0&0&0&0&0\\ 1&0&0&1&1&0&1&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&1&0&0\\ 1&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{pmatrix}.$$

Let $l = (l_1, l_2, l_3, l_4)$, $r = (r_1, r_2, r_3, r_4)$ be representations of field elements as vectors. Using an intermediate variable p_1 we can represent α by the following equations:

$$\begin{aligned}
&\alpha_1(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) = r_1,\\
&\alpha_2(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) = l_2 \oplus r_4,\\
&\alpha_3(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) = l_2 \oplus r_3 \oplus r_4 = \alpha_2(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) \oplus r_3,\\
&\alpha_4(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) = l_1 \oplus l_2 \oplus l_3 \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_4\\
&\ = \alpha_2(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) \oplus \alpha_5(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) \oplus l_3 \oplus r_2,\\
&\alpha_5(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) = l_1 \oplus r_1 \oplus r_3 = l_1 \oplus p_1,\\
&\alpha_6(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) = l_2 \oplus r_2,\\
&\alpha_7(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) = l_4 \oplus r_1 \oplus r_3 = l_4 \oplus p_1,\\
&\alpha_8(l_1, l_2, l_3, l_4, r_1, r_2, r_3, r_4) = l_3,\\
&p_1 = r_1 \oplus r_3.
\end{aligned}$$

Overall complexity of α is 9 XOR operations. In the same way, the complexity of ω is 5 XOR operations. So we get 14 Boolean operations to represent two linear mappings.

3.3. Elimination of branching

Let's return to item 2 of BPU algorithm: «2. If $r = 0$ then $l := \nu_0(l)$ else $l := \nu_1(l \circledcirc I(r))$». Write down a Boolean function which takes the value 1 at a single point $r = (0,0,0,0)$ and has zero value at all other points [4, p. 25–30]:

$$I_{0,0,0,0}(r) = \bar{r}_1 \cdot \bar{r}_2 \cdot \bar{r}_3 \cdot \bar{r}_4 = \overline{r_1 + r_2 + r_3 + r_4}.$$

The complexity of this function is 4 Boolean operations, the complexity of its negation — 3 operations. Now we can express l by a non-branching formula:

$$l^i = I_{0,0,0,0}(r) \cdot \nu_0^i(l) + \overline{I_{0,0,0,0}(r)} \cdot \nu_1^i(l \circledcirc I(r)), \quad i = 1, \ldots, 4. \tag{1}$$

It should be stressed that we compute $I_{0,0,0,0}(r)$ only once. To be more precise, we first calculate its negation (3 operations) and then, by double-negation law, compute $I_{0,0,0,0}(r)$ itself, which gives one additional operation. One needs 16 Boolean operations to represent four equations above. Using the fact that 0 is a fixed point for the permutation I we can reduce the amount of Boolean operations to 13.

Let's consider the following equations:

$$l^i = I_{0,0,0,0}(r) \cdot \big(\nu_0^i(l) \oplus \nu_1^i(0)\big) \oplus \nu_1^i(l \circledcirc I(r)), \quad i = 1, \ldots, 4. \tag{2}$$

If $r \neq (0,0,0,0)$, then

$$\overline{I_{0,0,0,0}(r)} \cdot \nu_1^i(l \circledcirc I(r)) = \nu_1^i(l \circledcirc I(r)), \quad i = 1, \ldots, 4,$$

and equations (1) and (2) are equivalent.

Otherwise, let r be equal to 0. The equation (1) is equivalent to

$$I_{0,0,0,0}(r) \cdot \nu_0^i(l) = \nu_0^i(l).$$

Using the fact that $I(0) = 0$ and $x \odot 0 = 0$ for all $x \in GF\,(2^4)$ it's rather easy to show that (2) is also equivalent to $\nu_0^i(l)$ for any $i = 1, \ldots, 4$:

$$\begin{aligned} I_{0,0,0,0}(r) \cdot \big(\nu_0^i(l) \oplus \nu_1^i(0)\big) \oplus \nu_1^i(l \circledcirc I(r))\big|_{r=0} \\ = I_{0,0,0,0}(0) \cdot \big(\nu_0^i(l) \oplus \nu_1^i(0)\big) \oplus \nu_1^i(l \circledcirc I(0)), \\ = 1 \cdot \big(\nu_0^i(l) \oplus \nu_1^i(0)\big) \oplus \nu_1^i(0) = \nu_0^i(l), \quad i = 1, \ldots, 4. \end{aligned} \tag{3}$$

At the same time $\nu_1^i(0) = (0, 0, 1, 0)$. That means that we only have to add one additional XOR operator to represent the equation (2).

After branching elimination the complexity of our representation has increased by 13 operations. The number of operations of each kind is given in a table below:

	AND	OR	NOT	XOR	Total
$\overline{I_{0,0,0,0}(r)}$		3			3
$I_{0,0,0,0}(r)$			1		1
Final glue by every coordinate	1			1(2)	
Final glue for all coordinaetes	4			5	9
				Total	13

4. Computer implementation of non-linear elements

4.1. Formulation of the problem

Consider a transformation table $2^4 \to 2^4$ or $2^8 \to 2^8$. The goal is to represent it as a set of Boolean functions in the basis of logical functions AND, OR, NOT, XOR with the minimal number of operations. The number of 1- and 2-input operations (AND, OR, NOT, XOR) required to implement a function will be used as the evaluation criteria.

This estimate coincides with the actual complexity of the circuit when it is implemented by integrated circuits containing corresponding logic elements. It also coincides if the FPGA function is used to implement it, since logical operations in this situation are formed using a matrix of macrocells, and the operation does not depend on the FPGA choice.

The algorithm is constructed as follows: at the first optimization step, the Quine–McCluskey algorithm is used, then the steps described below in sections 4.3.1-4.3.6 are iteratively repeated. After that, rules 4.4-4.7 are applied, with repetitive calls to procedures 4.3.1-4.3.6 if necessary.

For an example illustrating the steps of the algorithm we use the table of values of the ν_1 function from BPU-decomposition:

Table 1

X	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Y	7	6	c	9	0	f	8	1	4	5	b	e	d	2	3	a

Each of the Y bits may be represented in the form of PDNF, consisting of precisely 8 members since it is a coordinate function of a permutation. For example, for Y_0 in this example, this function will have the following form:

$$\begin{aligned}Y_0 = (\overline{X}_0 \cdot \overline{X}_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_0 \cdot X_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_0 \cdot \overline{X}_1 \cdot X_2 \cdot \overline{X}_3) \\ + (X_0 \cdot X_1 \cdot X_2 \cdot \overline{X}_3) + (X_0 \cdot \overline{X}_1 \cdot \overline{X}_2 \cdot X_3) + (\overline{X}_0 \cdot X_1 \cdot \overline{X}_2 \cdot X_3) \\ + (\overline{X}_0 \cdot \overline{X}_1 \cdot X_2 \cdot X_3) + (\overline{X}_0 \cdot X_1 \cdot X_2 \cdot X_3).\end{aligned}$$

The complexity of this representation is 47 operations.

Using the optimization techniques shown below, this function may be simplified to the following form:

$$Y_0 = \overline{(X_1 + X_2) \oplus X_0 \oplus X_3}.$$

The complexity of this representation is 4. Thus, for this particular example, the complexity decreases by more than 11 times as a result of our optimization.

As a second example, consider Y_3:

$$\begin{aligned}Y_3 = (\overline{X}_0 \cdot X_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_0 \cdot X_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_0 \cdot \overline{X}_1 \cdot X_2 \cdot \overline{X}_3) \\ + (\overline{X}_0 \cdot X_1 \cdot X_2 \cdot \overline{X}_3) + (\overline{X}_0 \cdot X_1 \cdot \overline{X}_2 \cdot X_3) + (X_0 \cdot X_1 \cdot \overline{X}_2 \cdot X_3) \\ + (\overline{X}_0 \cdot \overline{X}_1 \cdot X_2 \cdot X_3) + (X_0 \cdot X_1 \cdot X_2 \cdot X_3).\end{aligned}$$

This function representation requires 45 operations.

By means of optimization this function may be simplified to the following form:

$$Y_3 = ((X_0 \oplus X_3) \cdot X_2) \oplus X_1.$$

The complexity of this representation of the function is 3, which is 15 times better than the original one.

4.2. Quine–McCluskey algorithm

One of the algorithms for minimizing Boolean functions was proposed by Willard Quine and improved by Edward McCluskey. Since the algorithm is described in sufficient detail in many sources (see, for example, [5, p. 232–239]), its implementation will not be described in this paper.

Consider the simplification of the above examples of functions with this algorithm. For a formula representing the function Y_0 (with initial complexity of 47 operations)

$$\begin{aligned}Y_0 = (\overline{X}_0 \cdot \overline{X}_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_0 \cdot X_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_0 \cdot \overline{X}_1 \cdot X_2 \cdot \overline{X}_3) \\ + (X_0 \cdot X_1 \cdot X_2 \cdot \overline{X}_3) + (X_0 \cdot \overline{X}_1 \cdot \overline{X}_2 \cdot X_3) + (\overline{X}_0 \cdot X_1 \cdot \overline{X}_2 \cdot X_3) \\ + (\overline{X}_0 \cdot \overline{X}_1 \cdot X_2 \cdot X_3) + (\overline{X}_0 \cdot X_1 \cdot X_2 \cdot X_3)\end{aligned}$$

after optimization we get a formula with complexity 29 (1.5 times less complexity for this example):

$$\begin{aligned}Y_0 = (\overline{X}_0 \cdot \overline{X}_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_0 \cdot \overline{X}_1 \cdot \overline{X}_2 \cdot X_3) \\ + (X_0 \cdot X_1 \cdot \overline{X}_3) \\ + (X_0 \cdot X_2 \cdot \overline{X}_3) + (\overline{X}_0 \cdot X_1 \cdot X_3) + (\overline{X}_0 \cdot X_2 \cdot X_3).\end{aligned}$$

For the Y_3 function (the initial complexity of the representation is 45 operations), the Quine–McClusky method gives the result of 22 operations (optimization by 2 times):

$$\begin{aligned}Y_0 = (X_0 \cdot \overline{X}_1 \cdot X_2 \cdot \overline{X}_3) + (\overline{X}_0 \cdot \overline{X}_1 \cdot X_2 \cdot X_3) + (\overline{X}_0 \cdot X_1 \cdot \overline{X}_3) \\ + (X_0 \cdot X_2 \cdot X_3) + (X_1 \cdot \overline{X}_2).\end{aligned}$$

4.3. Further optimization of the Quine–McClaskey algorithm

As further steps for optimizing Boolean functions, an algorithm consisting of several steps is proposed. Moreover, each step is performed for all members of the polynomial inside the brackets. If optimization has been performed at any of the steps of the algorithm, then the algorithm starts again from the first step.

4.3.1. Duplicate Elimination

This step removes duplicates that may have appeared in previous iterations:

$$\begin{aligned}X_1 \cdot X_1 \cdot X_2 = X_1 \cdot X_2, \\ X_1 + X_1 + X_2 = X_1 + X_2, \\ X_1 \oplus X_1 \oplus X_2 = X_2.\end{aligned}$$

4.3.2. Removing the common factor under the logical *OR* performed on logical *AND*

$$(X_1 \cdot X_2) + (X_1 \cdot X_3) = X_1 \cdot (X_2 + X_3).$$

For the above example (Y_0 function with complexity 47), the representation after this transformation takes the form

$$Y_0 = \big(\overline{X}_0 \cdot ((\overline{X}_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_3 \cdot (X_1 + X_2)))\big) \\ + \big(X_0 \cdot \big((\overline{X}_1 \cdot \overline{X}_2 \cdot X_3) + ((X_1 + X_2) \cdot \overline{X}_3)\big)\big)$$

with the complexity value of 20.

4.3.3. Optimization of occurrences of the same term with and without negation for logical AND and OR, as well as optimization of constant members

We use formulas $X_1 \cdot \overline{X}_1 = 0, X_1 + \overline{X}_1 = 1, X_1 \cdot 0 = 0, X_1 + 1 = 1$.

4.3.4. Optimization of «*NOT*»

At this step, an attempt is made to reduce the number of NOT operations, using a single formula

$$\overline{X}_1 + \overline{X}_2 = \overline{X_1 \cdot X_2}.$$

For the above example (formula with complexity 47), the representation of the function after this operation will take the following form:

$$Y_0 = \big(\big(\overline{X_1 + X_2 + X_3} + (X_1 + X_2) \cdot X_3\big) \cdot \overline{X}_0\big) \\ + \big(\big((\overline{X}_1 \cdot \overline{X}_2 \cdot X_3) + ((X_1 + X_2) \cdot \overline{X}_3)\big) \cdot X_0\big)$$

with complexity 18.

4.3.5. Search on the table of pattern optimal functions

At this step, a table of optimal functions is used. The construction of the table is described below. A table of values of the function being optimized (or its parts) is generated. After that the values from this table are compared with the reference ones from the table of optimal functions. If they coincide, the pattern optimal function is used instead of the original one. At the moment, the table of optimal functions is constructed for functions of up to 4 variables.

To construct the table of optimal functions, the following approach was used (let us consider the case of compiling a table for three variables). First, functions are created within one operation (for each of the *AND*,

OR, XOR operations) by sequentially arranging the brackets and NOT operations, for example:

$$F = X_0 + X_1 + X_2,\ F = \overline{X_0 + X_1 + X_2},\ F = \overline{X}_0 + X_1 + X_2,$$

$$F = \overline{\overline{X}_0 + X_1 + X_2},\ F = \overline{X}_0 + \overline{X}_1 + X_2,\ F = \overline{\overline{X}_0 + \overline{X}_1 + X_2},$$

$$F = \overline{X_0 + X_1} + X_2,\ F = \overline{\overline{X_0 + X_1} + X_2},$$

and so on.

After that, formulas using combinations of binary operations are added (OR and XOR, for example), for which all combinations of parentheses and negation are also sorted out, for example:
$F = (X_0 + X_1) \oplus X_2, F = X_0 + (X_1 \oplus X_2), F = \overline{(X_0 + X_1)} \oplus X_2$, and so on.

At the next step, functions with tables of the same complexity are removed from the list of functions, while leaving the function with the minimum number of operations. The final list of pattern optimal functions is entered into the program in symbolic form from a file, which allows, on one hand, not to spend time on calculating the table each time the algorithm is run, and, on the other hand, it allows to supply the table with functions obtained by other algorithms or empirically.

We consider two examples of optimization of the components of the function Y_0

$$F = (\overline{X}_1 \cdot \overline{X}_2 \cdot X_3) + ((X_1 + X_2) \cdot \overline{X}_3).$$

The table of values for this function (the number of operations is 8) is the following:

X_1	0	1	0	1	0	1	0	1
X_2	0	0	1	1	0	0	1	1
X_3	0	0	0	0	1	1	1	1
F	0	1	1	1	1	0	0	0

In the table of optimal functions there is a function with complexity 2, the truth table of which is similar to the above:

$$F = X_3 \oplus (X_1 + X_2).$$

Accordingly, part of the polynomial may be replaced with this function.

Consider another example (the initial complexity of the representation is 8):

$$F = (\overline{X_1 + X_2 + X_3} + ((X_1 + X_2) \cdot X_3)) \cdot \overline{X}_0.$$

Note that in this polynomial there is a common part $X_1 + X_2$. Let us denote it by P_1. The function will take the form

$$F = (\overline{P_1 + X_3} + (P_1 \cdot X_3)) \cdot \overline{X}_0.$$

According to the truth table, the algorithm finds the following optimal function:

$$F = \overline{X_0 + (X_3 \oplus P_1)}.$$

Let us return from P back to $X_1 + X_2$

$$F = \overline{X_0 + (X_3 \oplus (X_1 + X_2))}.$$

The complexity of the resulting function is 4. As a result of this optimization step, the initial function will take the following form (complexity of 8 operations):

$$Y_0 = \left(\overline{(((X_1 + X_2) \oplus X_3) + X_0}\right) + \left(((X_1 + X_2) \oplus X_3) \cdot X_0\right).$$

4.3.6. Search for XOR possible usage

This step analyzes the possibility to use XOR, in accordance with the formulas

$$X_0 \cdot \overline{X}_1 + \overline{X}_0 \cdot X_1 = X_0 \oplus X_1,$$
$$\overline{(X_0 + X_1)} + (X_0 \cdot X_1) = \overline{X_0 \oplus X_1}.$$

For the above example, the formula for the function Y_0 will take the form

$$Y_0 = \overline{X_0 \oplus (X_1 + X_2) \oplus X_3}.$$

As a result, the complexity of the formula has become 4, which is about 10 times less than the original.

4.4. Additional optimization by combining brackets

In some cases, the above optimizations do not provide the minimal representation of the function. Consider an example. After a number of optimizations Y_2 function has become like that:

$$Y_2 = (\overline{X}_1 \cdot \overline{X}_2 \cdot X_3) + ((X_1 + X_2) \cdot \overline{X}_3).$$

An additional step is provided for handling such cases. If there are more than two terms under the brackets, then several functions are formed with different placement of brackets:

$$Y_2 = (\overline{X}_1 \cdot (\overline{X}_2 \cdot X_3)) + ((X_1 + X_2) \cdot \overline{X}_3), Y_2 = (\overline{X}_2 \cdot (\overline{X}_1 \cdot X_3)) + ((X_1 + X_2) \cdot \overline{X}_3),$$

$$Y_2 = (X_3 \cdot (\overline{X}_1 \cdot \overline{X}_2)) + ((X_1 + X_2) \cdot \overline{X}_3).$$

After that every formula from the set is optimized by the algorithm described above, and the formula with smallest resulting complexity is chosen. In our case it is the third function. It is transformed like this:

$$Y_2 = (X_3 \cdot (\overline{X}_1 \cdot \overline{X}_2)) + ((X_1 + X_2) \cdot \overline{X}_3) = (X_3 \cdot \overline{(X_1 + X_2)}) + ((X_1 + X_2) \cdot \overline{X}_3 \\ = (X_1 + X_2) \oplus X_3.$$

4.5. Primary XOR-optimization

In some cases, the optimization techniques discussed above do not lead to the best representation. Consider bit Y_3 of table 2:

$$Y_3 = (\overline{X}_0 \cdot X_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_0 \cdot X_1 \cdot \overline{X}_2 \cdot \overline{X}_3) + (X_0 \cdot \overline{X}_1 \cdot X_2 \cdot \overline{X}_3) \\ + (\overline{X}_0 \cdot X_1 \cdot X_2 \cdot \overline{X}_3) + (\overline{X}_0 \cdot X_1 \cdot \overline{X}_2 \cdot X_3) + (X_0 \cdot X_1 \cdot \overline{X}_2 \cdot X_3) \\ + (\overline{X}_0 \cdot \overline{X}_1 \cdot X_2 \cdot X_3) + (X_0 \cdot X_1 \cdot X_2 \cdot X_3).$$

The initial number of operations in the representation of this function is 45.

As a result of the above optimizations, one can get a function of the following form:

$$Y_3 = \big(\overline{(X_0 \oplus X_3) \cdot X_2} \cdot X_1\big) + \big((X_0 \oplus X_3) \cdot \overline{X}_1 \cdot X_2\big).$$

The number of operations in this representation is 9, which is 5 times less than the original.

Nevertheless, for representations of the function with the number of operations greater than 2, the algorithm makes an additional optimization attempt.

4.6. Using the Algebraic Normal Form (ANF)

Further, the program uses the representation of the function in the form of Zhegalkin polynomial (ANF). First, if one or more variables are found in the Zhegalkin polynomial in the form of terms of the first degree and are not found in any monomial anymore, they may be separated as terms modulo two, immediately reducing the dimension of the problem. If there are no such variables, in some cases it is still possible to reduce the complexity of the representation by separating certain variables as modulo two summands. Let us return to our example from section 3.1 (Table 1). Since the XOR function takes the value 1 when arguments are different and 0 when their values are the same, we can represent the value Y in the following form:

$$Y_i = X_j \oplus F(X),$$

where X_j is one of X bits, and $F(X)$ is equal to 0 for all cases where $Y_i = X_j$ and is equal to 1 if $Y_i \neq X_j$.

Obviously, one has to choose the bit X such that as many as possible $Y_i = X_j$, because in this case $F(X)$ will have less terms. Consider the values of Y_3 in the following table:

$\mathbf{X}_3$	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$\mathbf{X}_2$	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
$\mathbf{X}_1$	0	0	1	1	0	$\boxed{0}$	1	$\boxed{1}$	0	0	1	1	$\boxed{0}$	0	$\boxed{1}$	1
$\mathbf{X}_0$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$\mathbf{Y}_3$	0	0	1	1	0	$\boxed{1}$	1	$\boxed{0}$	0	0	1	1	$\boxed{1}$	0	$\boxed{0}$	1

For each of $X_i, i = 0, \ldots, 3$, the number of positions in which the value of X_i coincides with the value of Y, is calculated. For this example, we get the following result

$\mathbf{X}_0$	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$
8	12	8	8

Accordingly, the function will have the form

$$Y_3 = X_1 \oplus F(X).$$

We represent $F(X)$ in the form of PDNF with a value of 1 for cases of mismatch between X and Y (mismatching values are indicated by squares around them). So we get the function

$$\begin{aligned} Y_3 = X_0 \oplus \big((X_0 \cdot \overline{X}_1 \cdot X_2 \cdot \overline{X}_3) + (X_0 \cdot X_1 \cdot X_2 \cdot \overline{X}_3) \\ + (\overline{X}_0 \cdot \overline{X}_1 \cdot X_2 \cdot X_3) + (\overline{X}_0 \cdot X_1 \cdot X_2 \cdot X_3)\big). \end{aligned}$$

The complexity of this representation of the function is 21. However, $F(X)$ may be optimized by the methods indicated above. After the specified processing, the function takes the form

$$Y_3 = X_0 \oplus ((X_0 \oplus X_3) \cdot X_2).$$

The complexity of this representation is 3.

4.7. Merging common parts

As the final step, the algorithm attempts to find common parts both within one function and among all functions of the set, in order to optimize their calculation.

For example, for the above table, the algorithm worked out the following set of functions:

$$\begin{aligned}
Y_0 &= \overline{X_0 \oplus (X_1 + X_2) \oplus X_3},\\
Y_1 &= \overline{((X_2 + X_3) \oplus (X_0 \cdot X_2)) + X_1} \oplus (X_1 \cdot X_3),\\
Y_2 &= ((X_1 \oplus X_2) \cdot (X_0 \oplus X_3)) \oplus \overline{X}_2,\\
Y_3 &= X_0 \oplus ((X_0 \oplus X_3) \cdot X_2).
\end{aligned}$$

Total complexity is 19 operations. At the last step, the algorithm explores what is included in Y_0, Y_2, and Y_3. The algorithm denotes such common parts by the symbol P. For the given example, the following formulas are obtained:

$$\begin{aligned}
Y_0 &= \overline{P_0 \oplus (X_1 + X_2)},\\
Y_1 &= \overline{((X_2 + X_3) \oplus (X_0 \cdot X_2)) + X_1} \oplus (X_1 \cdot X_3),\\
Y_2 &= ((X_1 \oplus X_2) \cdot P_0) \oplus \overline{X}_2,\\
Y_3 &= X_0 \oplus ((P_0 \cdot X_2)),\\
P_0 &= (X_0 \oplus X_3).
\end{aligned}$$

After merging the common parts into separate functions, the complexity of our example is reduced from 19 to 17.

4.8. Optimization results

As a result of the above algorithm, the following set of Y functions was produced for the test data table:

$$\begin{aligned}
Y_0 &= \overline{P_0 \oplus (X_1 + X_2)},\\
Y_1 &= \overline{((X_2 + X_3) \oplus (X_0 \cdot X_2)) + X_1} \oplus (X_1 \cdot X_3),\\
Y_2 &= ((X_1 \oplus X_2) \cdot P_0) \oplus \overline{X}_2,\\
Y_3 &= X_0 \oplus ((P_0 \cdot X_2)),\\
P_0 &= (X_0 \oplus X_3).
\end{aligned}$$

The total complexity of this set of functions is 17, which is approximately 10 times less than the complexity of the set of non-optimized functions, equal to 188. The following are explicit formulas for all nonlinear functions involved in BPU-decomposition:

F	Presentation	Number of operations
I	$Y_0 = \left(X_0 \cdot \overline{X}_1\right) + \left(\left((X_0 \oplus X_1) + \overline{P}_1\right) \cdot X_3\right)$ $Y_1 = (X_0 \cdot X_2) \oplus \left(\overline{X}_1 \cdot P_1 \cdot P_2\right) \oplus X_3$ $Y_2 = ((X_1 \oplus X_3) \cdot ((X_0 + X_2) \oplus X_1)) \oplus X_2$ $Y_3 = \left(X_1 \cdot \overline{X}_2\right) + (((X_1 \oplus X_2) + P_2) \cdot X_0)$ $P_1 = X_0 \oplus X_2$ $P_2 = X_2 \oplus X_3$	26
v_0	$Y_0 = \left(X_1 \cdot \overline{X}_2\right) + \left(\left((X_1 \oplus X_2) + \overline{X_1 \oplus X_3}\right) \cdot X_0\right)$ $Y_1 = \left(\overline{(X_0 \oplus X_2) \cdot X_3} \cdot X_1\right) + \overline{P}_1$ $Y_2 = (X_1 \cdot X_3) \oplus \left(\left((X_2 \oplus X_3) + \overline{X}_1\right) \cdot X_0\right) \oplus \left(X_2 \cdot \overline{X}_3\right)$ $Y_3 = ((X_1 \oplus P_1) \cdot X_2) \oplus ((X_0 \oplus X_3) \cdot X_1)$ $P_1 = X_0 + X_3$	29
v_1	$Y_0 = \overline{(X_1 + X_2) \oplus P_1}$ $Y_1 = \overline{((X_2 + X_3) \oplus (X_0 \cdot X_2)) + X_1} \oplus (X_1 \cdot X_3)$ $Y_2 = ((X_1 \oplus X_2) \cdot P_1) \oplus \overline{X}_2$ $Y_3 = (X_2 \cdot P_1) \oplus X_1$ $P_1 = X_0 \oplus X_3$	29
φ	$Y_0 = \left(\overline{\left(X_0 \cdot \overline{X}_1\right) + (X_1 \oplus X_3)} \cdot X_2\right) \oplus \left(\overline{X}_1 \cdot X_3\right) \oplus \overline{X}_0$ $Y_1 = \overline{((X_0 + X_3) \cdot X_1) + X_2} \oplus (X_2 \cdot X_3)$ $Y_2 = \left(\overline{X}_0 \cdot X_3\right) + \left(\left(\left(X_0 + \overline{X}_1\right) \oplus (X_0 \cdot X_3)\right) \cdot X_2\right)$ $Y_3 = (X_0 \cdot X_1) \oplus \left(\left(\left(X_2 + \overline{X}_3\right) \oplus (X_1 \cdot X_2)\right) \cdot \overline{X}_0\right)$	33
σ	$Y_0 = \left(X_0 \cdot \overline{X}_1\right) + \left(\overline{X_1 \cdot P_1} \cdot X_3\right)$ $Y_1 = (((X_2 + X_3) \oplus (X_1 \cdot X_2)) \cdot X_0) \oplus ((X_2 \oplus X_3) \cdot X_1) \oplus X_3$ $Y_2 = (X_0 \cdot X_1) \oplus \left(\left((X_0 \cdot X_3) \oplus \overline{X}_1\right) \cdot X_2\right) \oplus \overline{X}_1 \oplus X_3$ $Y_3 = \left(X_2 \cdot \overline{X}_3\right) + \left(\left(\overline{X}_3 + P_1\right) \cdot \overline{X}_1\right)$ $P_1 = X_0 \oplus X_2$	31

Summary

In this paper we consider the possibility of bit-slicing the non-linear bijective mapping of GOST R 34-12.2015 «Kuznyechik» block cipher. The «Kuznyechik» permutation, represented by BPU-decomposition, fits into 235 Boolean operations (Table 2) after program optimization of small non-linear elements and algebraic features of finite field multiplication.

Table 2

	AND	OR	NOT	XOR	Total
I	8	5	4	9	26
v_0	9	5	6	9	29
v_1	4	3	3	7	17
φ	11	6	8	7	32
σ	11	6	7	9	33
α				9	9
ω				5	5
multiplication in $GF\left(2^4\right)$	16			15	31
multiplication in $GF\left(2^4\right)$	16			15	31
branchng elimination	4	3	1	5	13
	79	28	29	90	Total: 226

It should be noted that in 2016 a method of constructing S-boxes with minimal number of logical elements got a patent in Russian Federation [6]. The method protected by this patent allows to realize non-linear mapping of «Kuznyechick» cipher with complexity of 681 operations (254 ANDs and 436 XORs).

Acknowledgement: The authors would like to thank Dmitry Pronkin for helpful discussions on results and writing of the paper.

References

[1] Federal Agency on Technical Regulating and Metrology, "GOST R 34.12-2015. National standard of Russian Federation. Block ciphers.", 2015 (in Russian).

[2] Biryukov A., Perrin L., Udovenko A., *Reverse-engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1*, Cryptology ePrint Archive, Report 2016/071, 2016, http://eprint.iacr.org/2016/071.

[3] https://is.muni.cz/el/1423/podzim2015/VPL405/59270121/59761464/Logic_Friday_1/?lang=en.

[4] Yablonsky S. V., *Introduction to Discrete Mathematics*, 4th edition, Moscow: Vysshaya Shkola, 2006 (in Russian), 384 pp.

[5] Savel'ev A. Y., *Introduction to Informatics*, Moscow: Bauman Univ. Publ., 2001 (in Russian), 328 pp.

[6] Borisenko N. P., Vasinev D. A., Khoang Dyk Tkho, "Method of forming S-blocks with minimum number of logic elements", Abstract of Invention, RU 2572423 C2, 2016 (in Russian).