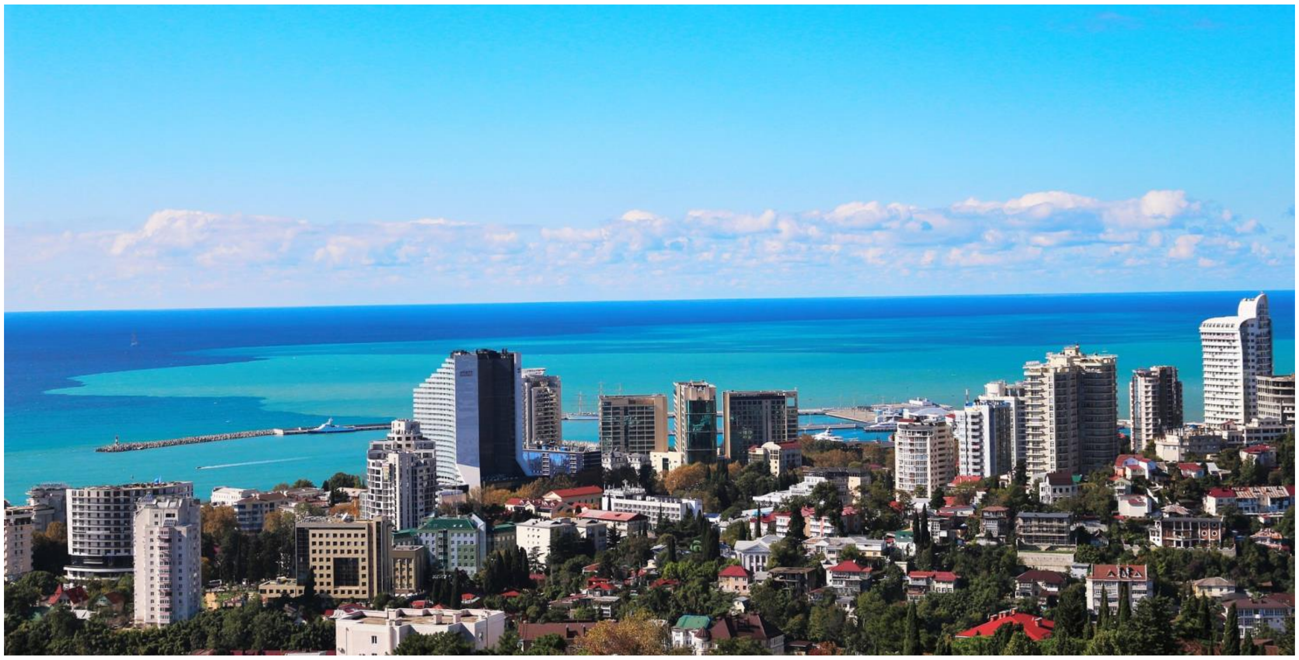
Proceedings of the 12th International Conference on Security of Information and Networks

September 12-15, 2019 Sochi, Russia



Organized by:

- Southern Federal University, Taganrog, Rostov-on-Don, Russia
- Sochi State University, Sochi, Russia
- Cardiff University, Cardiff, UK
- The University of Glasgow, Glasgow, UK
- Aksaray University, Aksaray, Turkey
- Rutgers University, NJ, USA
- Kennesaw State University, GA, USA
- Malaviya National Institute of Technology, Jaipur, India
- Manipal University Jaipur, Jaipur, India
- Macquarie University, Sydney, Australia

Edited by Oleg Makarevich, Ludmila Babenko, Maxim Anikeev,
Atilla Elçi & Hossain Shahriar

This publication is supported by the Russian Foundation for Basic Research (RFBR), grant No. 19-47-231001.

The Association for Computing Machinery
2 Penn Plaza, Suite 701
New York New York 10121-0701

ACM ISBN: 978-1-4503-7242-8

Table of Contents

Investigation of the Different Implementations for the New Cipher Qamal

Kunbolat Algazy
Institute of Information and Computational Technologies, Almaty, Kazakhstan
kunbolat@mail.ru

Rustem Biyashev
Institute of Information and Computational Technologies
Almaty, Kazakhstan

Nursulu Kapalova
Institute of Information and Computational Technologies
Almaty, Kazakhstan
kapalova@ipic.kz

Ludmila Babenko
Institute of Computer Technologies and Information Security of the Southern Federal University
Taganrog, Russia
lkbabenko@sfedu.ru

Evgeniya Ishchukova
Institute of Computer Technologies and Information Security of the Southern Federal University
Taganrog, Russia
uaishukova@sfedu.ru

Saule Nyssanbayeva
Institute of Information and Computational Technologies
Almaty, Kazakhstan
sultasha1@mail.ru

ABSTRACT

Currently, the Republic of Kazakhstan is creating a new standard for symmetric data encryption. Qamal encryption algorithm developed by the Institute of Information and Computer Technologies (Almaty, Republic of Kazakhstan), which is one of the candidates to be approved as a standard, is the subject of our study. We analyze in detail the basic cipher transforming work principles, approaches to its quick implementation and the results of the implementation experiments in several programming languages. The encryption algorithm under study uses the round subkeys generating procedure, which seems to be several times more complicated than the single block processing procedure.

The software implementation approaches suggested can significantly reduce computation time by using logic operations instead of accessing data arrays.

Our article is the first step to a comprehensive research of Qamal properties; its resistance to different cryptanalysis types is yet to be analyzed.

CSS CONCEPTS

• Security and privacy → Block and stream ciphers; Cryptanalysis and other attacks;

• Social and professional topics→ Computer science education;

KEYWORDS

Cryptography, Block Cipher, Standard, Secret Key, Implementation Speed, Logical Operations, Array Access

1 Introduction

People have always been interested in protecting their personal information. As soon as a number of different ciphers emerged, the countries started to think about the need to have a reliable data protection tool that would be proven to be persistent, have no loopholes and be used at various levels of trust.

The first standard to become known was DES [1]. Although the DES algorithm has been accepted for five years, it has maintained its standard status for more than 20 years and has never been compromised. However, the computing power had increased so much by 2000, that it became possible to pick up a secret key using powerful supercomputers of the time. DES was replaced by AES, a new cipher, that had been approved by the whole world, literally.

The GOST 28147-89 [2] standard of encryption is known to have been operated in Russia since 1989. It was classified till 1994, though. The new GOST R34.12-2015 encryption standard, which includes two ciphers, Magma and Kuznyechik [3], came into force in the Russian Federation in 2016.

The former Soviet Union countries are gradually switching to their own symmetric encryption standards. Thus, the STB 34.101.31-2007 "Information technologies and security. Cryptographic encryption and integrity control algorithms" [4] standard is used in the Republic of Belarus; and the standard based on the Kalina encryption algorithm was adopted in Ukraine in 2015 [5].

The Republic of Kazakhstan is also working on the symmetric data encryption state standard in the framework of the “Software and hardware development for cryptographic data protection in its transmission and storage in info communication systems and general purpose networks” targeted funding program of the Ministry of Education and Science of the Republic of Kazakhstan Committee of Science[13]. The Qamal cipher, proposed for the research in this article is one of the design encryption algorithms. Nowadays, the symmetric encryption algorithms being developed are subject to several requirements, including requirements for key parameters, strength, and software and hardware cipher implementations. Even though during the AES competition all the applicants were tested according to various evaluation criteria for software implementations, computing technology is developing at an amazing rate. That is why the research on the high-speed implementation remains actual, which has been confirmed by a few implementation studies and experimental tests for the AES cipher.

Thus, the authors presented the results for the use of the high-performing Cell broadband engine and NVIDIA graphics processing units research in [6].

In [7] they considered a version of high-speed implementation of the AES standard using Xilinx system generator implemented on Nexys-4 DDR FPGA development board and simulated using MATLAB Simulink. A 64-bit FPGA implementation of the 128-bit block and 128-bit-key AES cipher was presented in [10].

All this indicates the increased public interest the high-speed modern ciphers implementation[8, 9, 11, 12]. That is why our article is aimed at studying the Qamal cipher and its properties when performing various software implementations using different encryption algorithms.

The paper is organized as follows. The first chapter describes the Qamal project cipher, provides examples of the basic cipher transformations implementation that can be used as control points when performing a software cipher implementation. The second chapter discusses approaches to creating a high-speed implementation for the Qamal cipher. The third chapter presents the main experimental results obtained using different approaches to implementation and various programming languages.

2 Qamal encryption algorithm

2.1 Data encryption using Qamal's encryption algorithm

The Qamal encryption algorithm is a symmetrical block encryption algorithm, built on the SP-network principle. The algorithm supports block and key lengths of 128, 192 and 256 bits, while the length of the processed block of data and the length of the secret key must always match. The number of encryption rounds depends on the length of the block and the key. The 128-, 192- and 256-bit K keys correspond to eight, ten and twelve rounds of encryption, respectively. All rounds except the last ones are identical. In the last round, an additional round key is added. The scheme of the encryption algorithm is shown in Figure 1.

The encryption algorithm includes developed key imposition procedures using the bitwise addition (XOR) operation, the S-block replacement, the mixing procedures Mixer1 and Mixer2.

In the first procedure, a key modulo 2 operation (XOR operation) is performed in the plaintext block.

The second procedure to be produced is the bytes using the S-box replacement. For this, a nonlinear conversion of bytes is performed: a nonlinear bijective substitution is applied to each byte. The resulting S-box is presented in Table 1. In Table 1, all data is given in hexadecimal. Using an S-block here is similar to using an S-block for the AES data encryption standard: the original byte is divided into two halves, the top 4 bits (left side of the byte) indicate the row number in the replacement table, and the bottom 4 bits (right side of the byte) indicate the column number in the replacement table.

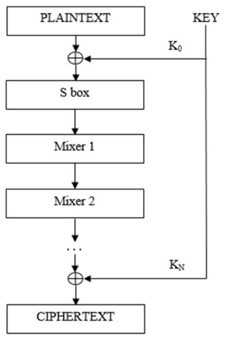


Figure 1 – Qamal Encryption Algorithm Scheme

The third procedure is the linear formation of the Mixer1 block. Bytes of the block are represented in the form of a two-dimensional array A with the size m*4, where m takes on the value of 4, 6 or 8, depending on the size of the initial block:

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ . & . & . & . \\ a_{m0} & a_{m1} & a_{m2} & a_{m3} \end{bmatrix}.$$

Bytes of each column are added together modulo 256:

$$M_1(b_{ij}) = \sum_{i-0}^{m} a_{ij} \ mod \ 256, j = 0,1,2,3.$$

Then, the received new byte of the first column replaces of the upper byte a_{00}, and the original bytes are shifted down by one position. This operation is repeated four times. As a result, we get four new bytes in the first column. Further, this operation is performed for the three remaining columns (Figure 2).

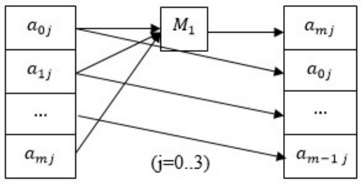


Figure 2 – Operation of the Mixer1

Table 1 – S1-block

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	c9	34	f0	18	55	86	21	6b	87	d2	6e	99	bd	31	98	89
1	29	73	83	8b	1a	19	e1	e4	f3	5b	72	3f	a6	f9	2e	a3
2	7e	10	94	07	ec	ad	2f	26	20	93	66	3d	dd	64	5f	c1
3	13	e0	80	25	d3	08	75	6a	b9	2d	d1	cc	fd	ca	3b	fc
4	d5	da	e2	ce	a0	7f	ae	c8	9c	09	3c	95	ba	35	3e	7b
5	fa	8d	23	ab	d9	e8	74	2a	c3	a8	d8	52	45	b5	0a	0c
6	a4	61	9a	fb	aa	f6	78	84	c4	e9	ee	54	50	81	df	90
7	36	b4	bb	44	c5	96	4b	28	14	e6	8f	ff	b0	1f	53	47
8	00	4c	40	2c	9b	9f	4a	01	7d	af	92	56	7a	db	8e	16
9	63	24	a9	1d	33	4d	e7	1c	70	69	b7	c6	32	e5	57	03
a	97	a5	eb	d4	bc	5d	f8	85	06	f2	59	f4	17	22	38	dc
b	0b	fe	be	cd	41	82	04	0e	48	71	30	ac	ef	c7	2b	cb
c	b8	8c	5a	42	a7	4e	d0	46	bf	b3	91	e3	11	7c	6f	de
d	88	58	1e	5c	9d	60	c0	62	05	79	ed	76	c2	02	65	d7
e	f1	8a	77	f7	37	b1	0f	67	cf	0d	a1	6c	4f	3a	39	1b
f	27	b6	5e	f5	ea	6d	15	9e	b2	12	a2	68	43	51	49	d6

The formation of the Mixer1 block results in a new B array of m*4 size, where m equals 4, 6 or 8 depending on the block size (m=4 for a block of 128 bits, m=6 for a block of 192 bits and m=8 for a block of 256 bits, respectively):

$$B = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ . & . & . & . \\ b_{m0} & b_{m1} & b_{m2} & b_{m3} \end{bmatrix}.$$

Each row of the array is represented as a polynomial and a polynomial multiplication of the form $b_i(x)$ * $m_i(x)$ mod p(x) occurs, where:

p = 1000000000000000000000000001001101112 = 0x100000137;
m_0 = 10101000001000101011101110111010 2 = 0xA822BBBA;
m_1 = 110100100011010111010010011001012 = 0xD235D265;
m_2 = 110110100001100110010110110100102 = 0xDA1996D2;
m_3 = 100100000100101110011110000110112 = 0x904B9E1B;
m_4 = 101000110000010001101111011010102 = 0xA3046F6A;
m_5 = 100101101110110100001101001101012 = 0x96ED0D35;
m_6 = 011000110011101101101000110011012 = 0x633B68CD;
m_7 = 101001110011000111110001100110102 = 0xA731F19A.

The $m_i(x)$ values are also presented as polynomials and are applied as follows. With an open block length of 128 bits, the first four values of $m_0(x)$, $m_1(x)$, $m_2(x)$, $m_3(x)$ are used. For the block length of 192 bits the first 6 six values of $m_0(x)$, $m_1(x)$, $m_2(x)$, $m_3(x)$, $m_4(x)$, $m_5(x)$ are taken. For the third possible block length, all the eight $m_i(x)$ values are used.

2.2 Data decryption using Qamal's encryption algorithm

To decrypt the ciphertext, all cryptographic transformations used for encryption are inverted and exercised in the decryption algorithm in reverse order. Round keys are also used in reverse order. When decrypting each of these block lengths, 8, 10, or 12 rounds are performed, respectively, with InvS, InvM1, and InvM2 inverse conversions performed in each block length.

The InvS conversion is inverse to the byte change operation through the S-block in Table 1. For example, if the 00 byte is replaced by the C9 byte, the C9 byte is to be replaced by the 00 byte for inverse conversion.

The InvM1 transform is the reverse of the Mixer1 transform. That means that the additional operation 256 modulo must be replaced by a sequential subtraction operation modulo 256.

The conversion of InvM2 is inverse to the procedure for obtaining a Mixer2 block. To obtain a block inverse Mixer2, each row of the array is treated as a polynomial, which is multiplied by fixed modulo p (x) polynomials, where:

p = 1000000000000000000000000001001101112 = 0x100000137;
m_0^{-1} = 1111 0011 0100 1000 1000 1001 1101 01012 = 0xf3488ad5;
m_1^{-1} = 0001 0110 0111 0011 1101 0000 1101 01112 = 0x1673d0d7;
m_2^{-1} = 1000 1010 0010 1110 1000 1011 1011 10102 = 0x8a2e8bba;
m_3^{-1} = 1100 0000 1010 0010 0011 1100 1011 00002 = 0xc0a23cb0;
m_4^{-1} = 1111 1101 1010 0101 0110 0100 0101 00102 = 0xfda56452;
m_5^{-1} = 0111 1111 1001 1100 0011 0000 0010 00102 = 0x7f9c3022;
m_6^{-1} = 1001 1000 0100 1011 1001 1101 0011 11102 = 0x984b9d3e;
m_7^{-1} = 0001 1110 0111 0011 0001 1111 1000 10002 = 0x1e731f88;

The first four values $m_0^{-1}(x)$, $m_1^{-1}(x)$, $m_2^{-1}(x)$, $m_3^{-1}(x)$ are used for an open block of 128 bits. For the block length of 192 bits the first 6 six values of $m_0^{-1}(x)$, $m_1^{-1}(x)$, $m_2^{-1}(x)$, $m_3^{-1}(x)$, $m_4^{-1}(x)$, $m_5^{-1}(x)$ are taken. For the third possible block length, all eight $m_i^{-1}(x)$ values are used.

2.3 Round key generation algorithm

Round keys K_i are generated from the key K cipher using the key expansion procedure. As a result, an array of round keys is formed, from which the required round key is then directly selected. The scheme for obtaining round keys is shown in Figure 3.

The original secret key K is the first round subkey. To generate the following secret key, ten rounds of conversions are performed: the replacement with the S-box that is different from what was used during encryption, the transformation Mixer1 (Figure 2) and the transformation Mixer2, which is similar to the transformation of the same name in the encryption procedure.

The last procedure of the round is called Module $p_i(x)$. It works as follows. Let $g_1(x)$, $g_2(x)$, ..., $g_s(x)$ – non-delivable binary polynomials used as working bases, where $G(x) = g_1(x)g_2(x)...g_s(x)$. The degree of the polynomial G(x) corresponds to the value $N = m_1 + m_2 + ... + m_s$ and is equal to the block length (i.e.128, 192, 256). The output from the Mixer2 block is represented as an N (x) polynomial with binary coefficients. $k_1(x)$, $k_2(x)$, ..., $k_s(x)$ – the remains of the division of the polynomial N(x) on the corresponding bases $p_i(x)$, i = 1,...,s. Where $p_i(x)$, i = 1,...,s are the secret elements of the key deployment procedure.

The values of the polynomials p(x) for the Module operation are secret, that is, in fact, they make up additional key information. For the purposes of this study, we use the following polynomial values as a secret element $p_i(x)$ to obtain round keys:

$p_1(x)$ = 100000000001010112 = 0x1002B;
$p_2(x)$ = 100000000001011012 = 0x1002D;
$p_3(x)$ = 100000000001110012 = 0x10039;
$p_4(x)$ = 100000000001111112 = 0x1003F;
$p_5(x)$ = 100000000010001112 = 0x10047;
$p_6(x)$ = 100000000010100112 = 0x10053;
$p_7(x)$ = 100000000100011012 = 0x1008D;
$p_8(x)$ = 100000000101111012 = 0x100BD;

After ten rounds of transformations, the result is added modulo two with the value of the round subkey from which the current subkey was generated, resulting in a new round key. To get the next round key, you need to re-run a cycle of ten transformations.

2.4 Data conversion example

Consider, for example, how message X will be encrypted on the K key, where:

X = 0x81754b8c671be306adee86fc52174dcd
K = 0x904b9e1bd6eaa64db9a9c168a5e5f92d

Let us start the process with the development of round subkeys. To generate the first round subkey, you must perform ten rounds of conversion. The initial value to which the transformation will be applied is the secret encryption K key = 0x904b9e1bd6eaa64db9a9c168a5e5f92d.

The first conversion is the replacement of bytes in accordance with Table 3. Thus, the high 0x90 byte is converted to 0x5, the next byte 0x4B to 0x2, and so on. As a result, the value K = 0xa5a2d21f5af9d99b18e364890a8503f1 is obtained. The next transformation is mixing the data using the Mixer1 conversion. In order to perform it, let us present the value K = 0xa5a2d21f5af9d99b18e364890a8503f1 as a square matrix, as it is shown in Table 2.

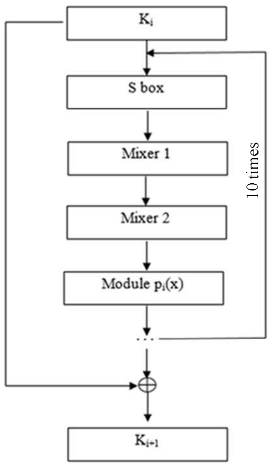


Figure 3 – Round key K_i expansion, where i = 0,1,...,8 (10,12)

Table 2 – Presentation of key data as a matrix

A5	A2	D2	1f
5a	F9	D9	9b
18	E3	64	89
0a	85	03	F1

We are working with the first (left) column of table 2. It is necessary to perform addition of all bytes modulo 256, write them to the topmost cell, and move the remaining cell values down by one. We need to repeat this action four times until all cells change their values (Table 3).

Let us perform the same actions with the remaining three columns and get a new state (Table 4).

As a result of Mixer 2 we get the following transformations:

(0x5645e32f * 0xa822bbba) mod 0x100000137 = 0x3c0d7d49
(0x581fde65 * 0xd235d265) mod 0x100000137 = 0xafeeb61e
(0x38812177 * 0xda1996d2) mod 0x100000137 = 0x6627b56f
(0x21031234 * 0x904b9e1b) mod 0x100000137 = 0x24c7ed8f

Collecting the results of conversions in one block, we get the result of the conversion Mixer2: K = 0x3c0d7d49afeeb61e6627b56f24c7ed8f.

Table 3 – Result of the first column conversion using Mixer1

The initial state	First change	Second change	Third change	Fourth change
a5	(a5+5a+ +18+0a) mod 256 = 21	(21+a5+5a+ +18) mod 256 = 38	(38+21+ +a5+5a) mod 256 = 58	(58+38+21+ +a5) mod 256 = 56
5a	a5	21	38	58
18	5a	a5	21	38
0a	18	5a	a5	21

Table 4 – Mixer1 conversion result

56	45	e3	2f
58	1f	de	65
38	81	21	77
21	03	12	34

The last round of conversion is the Module conversion. To perform it, the value K = 0x3c0d7d49afeeb61e6627b56f24c7ed8f must be sequentially taken over eight different modules and a new block must be created from the obtained values:

0x3c0d7d49afeeb61e6627b56f24c7ed8f mod 0x1002b = 0x0ccf
0x3c0d7d49afeeb61e6627b56f24c7ed8f mod 0x1002d = 0x5463
0x3c0d7d49afeeb61e6627b56f24c7ed8f mod 0x10039 = 0xb236
0x3c0d7d49afeeb61e6627b56f24c7ed8f mod 0x1003f = 0xa02f
0x3c0d7d49afeeb61e6627b56f24c7ed8f mod 0x10047 = 0x338f
0x3c0d7d49afeeb61e6627b56f24c7ed8f mod 0x10053 = 0xd5c2
0x3c0d7d49afeeb61e6627b56f24c7ed8f mod 0x1008d = 0x7714
0x3c0d7d49afeeb61e6627b56f24c7ed8f mod 0x100bd = 0xf02f

By combining the obtained results, we get the result of the Module conversion: K = 0x0ccf5463b236a02f338fd5c27714f02f.

This completes the conversion of the first round. The further nine rounds of conversion are required to obtain the second round key. Each round consists of four operations: replacing with the help of S-block, transformation Mixer1, transformation Mixer2, transformation Module. Transformations from the second to the tenth round are presented in table 5.

In order to get the result of the second round subkey, we need to add the result from table 5 modulo two with the value of the first round subkey K = 0x904b9e1bd6eaa64db9a9c168a5e5f92d. Thus, we get:

K = 0x270d01b891fbd94d7433cad069ba874d ⊕ 0x904b9e1bd6eaa64db9a9c168a5e5f92d = = 0xb7469fa347117f00cd9a0bb8cc5f7e60

As a result of all the transformations, we get nine round subkeys, presented in Table 6.

Once the round subkeys have been developed, you can start encrypting the text. As a test vector, which should be encrypted, the value Text = 0x81754b8c671be306adee86fc52174dcd was chosen. The first transformation of the round is the addition of data from the first subkey round:

Text = 0x81754b8c671be306adee86fc52174dcd ⊕ 0x904b9e1bd6eaa64db9a9c168a5e5f92d = 0x113ed597b1f1454b14474794f7f2b4e0.

Table 5 – Developing a second round subkey (rounds 2-10)

Operation	Result
S-box:	K = 0x278ab713d1708dacc7d7569fab7bc5ac
Mixer1:	K = 0x4556abb68b639c31291df9686a4c5f0a
Mixer2:	K = 0x7c6df73a33debe79be69fa62595adff8
Module P 1	K = 0x4f83630ba1bb2d74bab883a5adc0532f
S-box:	K = 0xfd151351c4ccf1f2e083150a0fe1ebac
Mixer1:	K = 0xc0d25912c2cf258251a91d46b04504f9
Mixer2:	K = 0x23520af35fb22e94774387edd0e2e37d
Module P 2	K = 0x2dffe89209aae42b90f41363fc93e67e
S-box	K = 0xf16c86a72cf053a6a5b6ce13f4fa6e16
Mixer1	K = 0x6a1c018c4b86aa99781ebcd6b60c1576
Mixer2	K = 0x5c4c8ca13c893231108af639d3a61920
Module P 3	K = 0x9f42de86ee6a1de488ff40fc656ea045
S-box:	K = 0x62300b8f2a73cb534a6c88f42fd88d35
Mixer1	K = 0xae8d49496c800acedbf649e105e7eb0b
Mixer2	K = 0x807c17bc323f3a64b0a07f48d23bc9e9
Module P 4	K = 0xc46a252a7f8eda5748168ee6a3c4f418
S-box	K = 0x4c73df900e7563afb484756e264cb6bf
Mixer1	K = 0x921343d9d0c4d3c44224241934b86d6c
Mixer2	K = 0x4a3278ae78d112bf629d456ff0dbf334
Module P 5	K = 0x18a1f69e77e8cb29739948b9f636282b
S-box	K = 0xbfc437d2ab86c157239db418377091a6
Mixer1	K = 0x53387b197fdf1e38513ee928c4573de7
Mixer2	K = 0x8190f1574c00ce5a5019f844d04a43a9
Module P 6	K = 0xbe2e0f5e481937a89d323c37ee83904c
S-box	K = 0xd541c2a9b4e6e9f758cb0e92a15a560
Mixer1	K = 0x5aee23ff876a06fb467b5b72b8c800e9
Mixer2	K = 0xeb45c1be3db33e4590ef823d3e0770db
Module P 7	K = 0xdc5165e3ecfbc1e18157f4b19704365f
S-box	K = 0x93c62f5b81d664c81afb6a361c370f3
Mixer1	K = 0xa93038ee15034e5b8b59027f760eb9b9
Mixer2	K = 0xccca3a6555d21719fd3d640d52286c23
Module P 8	K = 0x8406a687d232289ad32fefec9d0360a
S-box	K = 0xe7b1d907008c91322d8cf8f808e87009
Mixer1	K = 0x66444f8a336870de307a346b1cb1d23a
Mixer2	K = 0x3f342d47be4ec33d5f8367efad24ff02
Module P 9	K = 0x270d01b891fbd94d7433cad069ba874d

Table 6 – Received round subkeys

Subkey	Subkey value
K1	0x270d01b891fbd94d7433cad069ba874d
K2	0xb7469fa347117f00cd9a0bb8cc5f7e60
K3	0x756012f7ba128da67efba084be1b78cf
K4	0x411ded5410a74f8489d5a891ff54f91f
K5	0x7a574831d813d72a3360f34b30b9dc06
K6	0xe808de737d501937f397539aa152bbdf
K7	0xbf674c851c0bc3ddf5c043d0ca87b4b
K8	0x72a34ba12595f9cd1b629a163e546836
K9	0x3d173b7e2961a26293b178bd70c77460

After that we replace bytes using the S-box (Table 1). As a result, the data block will look as follows: Text = 0x733b601cfeb67f951ac8c8339e5e41f1.

The next transformation is to mix the data using the transformation Mixer 1. In order to perform it, let us present the value Text = 0x733b601cfeb67f951ac8c8339e5e41f1 as a square matrix, as it is shown in Table 7. The result of the Mixer1 conversion is presented in Table 8.

Table 7 – Presentation of Text data as a matrix

73	3b	60	1c
fe	b6	7f	95
1a	c8	c8	33
9e	5e	41	f1

Table 8 – Mixer1 conversion result

9e	fa	2d	e9
4e	d8	56	3f
b4	d0	8f	b9
29	17	e8	d5

If we write values from Table 8 line by line, we get the block value as a result of the conversion of Mixer1: Text = 0x9efa2de94ed8563fb4d08fb92917e8d5.

As a result of using the Mixer2 operation, we get the following transformations:

(0x9efa2de9 * 0xa822bbba) mod 0x100000137 = 0xe5d8e3a5
(0x4ed8563f * 0xd235d265) mod 0x100000137 = 0x5c899c10
(0xb4d08fb9 * 0xda1996d2) mod 0x100000137= 0x1a5323de
(0x2917e8d5 * 0x904b9e1b) mod 0x100000137 = 0xad001adf

Collecting the results of conversions in one block, we get the result of the conversion Mixer2: Text = 0xe5d8e3a55c899c101a5323dead001adf.

This completes the conversion of the first round of encryption. Eight rounds of encryption are used for the 128-bit block. All intermediate values from the second to the eighth rounds are shown in Table 9. You can see that the encryption results in encrypted text: Cipher = 0x2040844e82689d9279fd3bce5c67541.

Table 9 – Transformations of 2-8 rounds of encryption

R	Operation	Result
2	XOR K2	Text = 0x529e7c061b98e310d7c92866615f64bf
	S-box	Text = 0x2357b0213f70f72962b32078610caacb
	Mixer1	Text = 0xa12aa923704d5026e900384f2586718d
	Mixer2	Text = 0xa16ce2f0d6b462f612ab4b6ab5d8f998
3	XOR K3	Text = 0xd40cf0076ca6ef506c50ebee0bc38157
	S-box	Text = 0x9dbd276b50f81bfa50fa6c3999424c2a
	Mixer1	Text = 0x5c94ad2cd646e49313a0a866d6f1fac8
	Mixer2	Text = 0x95d27d43bf26a0c7b4ea6b2b072cd3e
4	XOR K4	Text = 0xd4cf9017af81ef43829b0e234f263421
	S-box	Text = 0x9dde63e4dc4c1bce40c698077b2fd310
	Mixer1	Text = 0x5864b12c9a5866fded0fff82341fe9c9
	Mixer2	Text = 0x8f572d43da96805539997736ae66389d
5	XOR K5	Text = 0xf50065720285577faf9847d9edfe49b
	S-box	Text = 0x6dc9f6bbf09f2a476e129b1f57d737c6
	Mixer1	Text = 0xe869549b6c84bff1edcbad082251f2e7
	Mixer2	Text = 0x8df913275ec121504f553da5389c1cc2
6	XOR K6	Text = 0x65f1cd5423913867bcc26e3f99cea71d
	S-box	Text = 0xf6b67cd90724b984ef5adffc696f85f9
	Mixer1	Text = 0x1f843d3093547b5a41d7adab55a39952
	Mixer2	Text = 0x9cb06a2ac7ea32eb92d5498464e0a5c4
7	XOR K7	Text = 0x97461ee2962a8ed64d894db96848de8f
	S-box	Text = 0x1cae2e77e7668ec035af3571c49c6516
	Mixer1	Text = 0x7fc424f63395595b34224766fc5f56be
	Mixer2	Text = 0xca6fc89095a99a73fc4df640e48cf1b3
8	XOR K8	Text = 0xb8cc8331b03c63bee72f6c56dad89985
	S-box	Text = 0x48112ce00bfdfb2b67c15074ed05699f
	Mixer1	Text = 0xab0dc1615b855ec661a3579da7d4e01e
	Mixer2	Text = 0x3f13333ac1472bbbb42eab0195010121
	XOR K9	Text = 0x2040844e82689d9279fd3bce5c67541

3 Approaches to obtaining high-speed cipher implementation

Since the speed of cipher implementation is important for analysis, it is necessary to consider ways to increase its efficiency. The C programming language was chosen to implement the cipher faster. In C the maximum size of the variable is limited to 64 bits. Since the data block exceeds this size for the Qamal algorithm, you need to allocate two to four variables (depending on the block size) for data storage. The Qamal encryption algorithm is constructed in such a way that it often operates with bytes, for example, in a replace using an S-box and in a Mixer1 transformation. In classic implementation, it would be logical to use arrays or lists, saving each byte separately and operating with them. But accessing an array is a resource-intensive operation which significantly reduces the implementation speed. Therefore, we decided to consider another way to perform operations in the Qamal cipher, based on the use of simple logical operations such as shifts, conjunctions and disjunctions. In this section, all the operations are considered in relation to a data block of 128 bits. To store a block of 128 bits, you must use two 64-bit unsigned variables (unsigned long type). In the formulas indicated below, we use operations to convert the key, both parts of which are in two variables: KL (most significant bits) and KR (least significant bits). This approach can be easily scaled on block transformations, including a block of greater length.

Let us consider the operation of changing bytes using S-block replacement. To allocate the next byte to be replaced from a large 64-bit variable, you should shift the content of the variable to the right so that the byte is in the smallest significant bits and then multiply it by the value 0xff. As a result of multiplication all the state bits except for the lower byte are zeroed and we get the value we need:

```
buf = (KL >> sdv) & 255;
buf1 = (KR >> sdv) & 255;
```

The shift value varies from 56 to 0 in increments of 8 (by the number of bits per byte). Then it is necessary to replace the byte according to the replacement table. In order not to complicate the logic of dividing a byte into two parts and defining the row and column of the replacement table, we can rewrite the replacement table as a one-dimensional array. For example, the S-box for key conversion can be represented as a one-dimensional array. Replaced bits must be returned to the position they were originally located. Therefore, the result of the replacement according to the table must be shifted to the left by the same number of positions and added to the new variable using the addition modulo two:

```
buf_KL = buf_KL ^ (S_key[buf] << sdv);
buf_KR = buf_KR ^ (S_key[buf1] << sdv);
```

Thus, these actions must be performed 8 times. Since two halves of the text are changed at once, the result will be a conversion of 16 bytes.

The next transformation is the linear mixing Mixer1. It is important to note that bytes are written to the array line by line and converted by column. So you can see that for the first column the first and fifth bytes from variable KL, and also the first and fifth bytes from variable KR are required. The second column requires the second and sixth bytes of the KL and KR variables. For the third column - the third and seventh bytes. And finally, for the fourth column, the fourth and eighth bytes. Receiving bytes is similar to how it was done for bytes in the S-block conversion. But now it is necessary to take two bytes from one variable, which are exactly 32 positions apart:

```
a1 = (KL >> sdv) & 255;
a2 = (KL >> (sdv - 32)) & 255;
a3 = (KR >> sdv) & 255;
a4 = (KR >> (sdv - 32)) & 255;
```

Let us see how the values of a1, a2, a3, a4 change when applying the Mixer1 transformation. To do this, we will refer to Table 10. You should bear in mind that Table 10 shows the values of the variables, not hexadecimal values, as was done earlier. Our task is to determine which variables form the result of the Mixer1 conversion. When adding, we omit the "mod 256", however, we should remember that in the Mixer1 transformation, addition is performed by modulo 256.

Table 10 – Result of converting a single column using Mixer1

Source state	First change	Second change	Third change	Fourth change
a_1	a_1+a_2+ $+a_3+a_4$	a_1+a_2+ $+a_3+a_4+$ $+a_1+a_2+$ $+a_3 =$ 2^*a_1+ $+2^*a_2+$ $+2^*a_3+$ $+a_4$	$2^*a_1+2^*a_2+$ $+2^*a_3+a_4+$ $+a_1+a_2+a_3+$ $+a_4+a_1+a_2 =$ $= 4^*a_1+$ $+4^*a_2+$ $+3^*a_3+$ $+2^*a_4$	$4^*a_1+4^*a_2+$ $+3^*a_3+2^*a_4+$ $+2^*a_1+2^*a_2+$ $+2^*a_3+a_4+$ $+a_1+a_2+a_3+$ $+a_4+a_1=$ $=8^*a_1+7^*a_2+$ $+6^*a_3+4^*a_4$
a_2	a_1	a_1+a_2+ $+a_3+a_4$	2^*a_1+ $+2^*a_2+$ $+2^*a_3+a_4$	$4^*a_1+4^*a_2+$ $+3^*a_3+2^*a_4$
a_3	a_2	a_1	$a_1+a_2+a_3+a_4$	$2^*a_1+2^*a_2+$ $+2^*a_3+a_4$
a_4	a_3	a_2	a_1	$a_1+a_2+a_3+a_4$

From Table 10, you can see that the final conversion of the Mixer1 transformation can be obtained immediately, bypassing the intermediate conversions. The only thing to keep in mind is that all conversions are performed modulo 256:

```
b1 = (8*a1+7*a2+6*a3+4*a4) % 256;
b2 = (4*a1+4*a2+3*a3+2*a4) % 256;
b3 = (2*a1+2*a2+2*a3+a4) % 256;
b4 = (a1+a2+a3+a4) % 256.
```

The result of the conversion should be placed back to the positions from which the values a1, a2, a3, a4 were originally taken. To do this, the new values should be shifted to the left by the same number of positions and the addition modulo two operation should be performed with the variable, in which the result of transformation is accumulated:

```
buf_KL = buf_KL ^ (b1 << sdv) ^ (b2 << (sdv-32));
buf_KR = buf_KR ^ (b3 << sdv) ^ (b4 << (sdv-32)).
```

4 Experimental results

Our first experiment was aimed at determining the time needed to encrypt a single data block and produce round subkeys as well as finding out the total time of generating round keys and encrypting a single data block. The algorithm operation speed was measured by performing the same transformations 10 000

times dividing the value by 10 000 then. The algorithm implementation was performed in VisualStudio 2017 by using the C programming language. The obtained results were compared by a program using the array data storage in Python.
To maintain the experimental integrity all the measurements were performed on a single PC having the following technical characteristics: Ryzen 5 1500x with 3.5 GHz frequency and 16 GB of RAM (DDR4 with a 2400 MHz frequency). Table 11 represents the experiment results.
Comparing the total cipher time for implementations in both C and Python shows us that the C implementation speed for a 128-bit data block is 21.5 times faster; it is also 44 times faster for a 192-bit block, and 75 times faster for a 256-bit block. It can be explained by the fact that the round subkeys development takes most of the time and increasing processed block dimension, we increase the number of encryption rounds, as well.
Comparing the round keys generating time with respect to the block encryption time, we can see that the round subkeys generating takes tens of times longer than one block encryption time for both implementations the generation time of round keys with respect to the block encryption time (Table 12).

Table 11 – Implementation Speed Comparison

Programming language	Block size (bits)	№ of rounds	Keygen speed, sec	Block encr, sec	Total work time, sec
C	128	8	0,00128	0.000026	0.001306
Python			0,02733	0.0007799	0.02811339
C	192	10	0,0016	0,0000325	0,0016325
Python			0.0708	0.0015828	0.07222892
C	256	12	0,00192	0,000039	0,001959
Python			0.14594	0.0026801	0.14823825

Table 12 – Key Generation and Data Block Encryption Time Comparison

Programming language	Block size (bits)	Number of rounds	Ratio of key generation time to data block encryption time
C	128	8	49,23
Python			35,044
C	192	10	49,2307
Python			44,73
C	256	12	50,01
Python			54,45

5 Conclusion

We have considered the project Qamal symmetric encryption algorithm, one of the candidates to become the standard for data encryption in the Republic of Kazakhstan. As it has been described in the study, the procedure of generating round subkeys, used in the encryption algorithm is several times more complicated processing a single data block. If we apply it to encrypting large files, there seems to be no problem as the once developed round subkeys can be then used for encryption. But when it comes to on-the-fly key generation, where the encryption speed is measured by the speed of generating round subkeys, we see that it takes several times longer than the encryption itself. The encryption is yet to be thoroughly tested using other cryptoanalytic attacks to fully verify its reliability.

ACKNOWLEDGMENTS

The study is supported by the "Software and hardware development for cryptographic data protection in its transmission and storage in infocommunication systems and general-purpose networks" targeted funding program of the Ministry of Education and Science of the Republic of Kazakhstan Committee of Science (registration number 0118PK01064).

REFERENCES

[1] History of DES. - http://www.umsl.edu/~siegelj/information_theory/projects/des.netau.net/des%20history.html
[2] GOST 28147-89: Encryption, Decryption and Message Authentication Code (MAC) Algorithms // https://tools.ietf.org/html/rfc5830
[3] GOST R 34.12–2015 «Information technology. Cryptographic data security. Block ciphers» // https://tc26.ru/en/standards/standards/gost-r/gost-r-34-12-2015-information-technology-cryptographic-data-security-block-ciphers.html
[4] Fault-based Attacks on the Bel-T Block Cipher Family // https://zerobyte.io/publications/2015-JP-belt.pdf
[5] A New Encryption Standard of Ukraine: The Kalyna Block Cipher - https://pdfs.semanticscholar.org/7771/8fbf6c2044b6f1aa2e66a1eda99121caa4da.pdf
[6] Dag Arne Osvik, Joppe W. Bos, Deian Stefan, and David Canrigh Fast Software AES Encryption// FSE 2010 - https://www.iacr.org/archive/fse2010/61470076/61470076.pdf
[7] Altaf O. Mulani and Pradeep B. Mane High-Speed Area-Efficient Implementation of AES Algorithm on Reconfigurable Platform // DOI: 10.5772/intechopen.82434 - https://www.intechopen.com/online-first/high-speed-area-efficient-implementation-of-aes-algorithm-on-reconfigurable-platform
[8] Roman-Oliynykov/ciphers-speed // https://github.com/Roman-Oliynykov/ciphers-speed
[9] Patrick Favre-Bulle Security Best Practices: Symmetric Encryption with AES in Java and Android Jan 6, 2018 https://proandroiddev.com/security-best-practices-symmetric-encryption-with-aes-in-java-7616beaaade9
[10] Liberatori, Minica & Otero, Fernando & Bonadero, J.C. & Moreira, J. (2007). AES-128 cipher. High speed, low cost FPGA implementation. 195 - 198. 10.1109/SPL.2007.371748.
[11] Lee, Wai Kong. (2014). High Speed Implementation of Symmetric Block Cipher on GPU. https://www.semanticscholar.org/paper/High-speed-implementation-of-symmetric-block-cipher-Lee-Goi/bbe2efba9e3c48034511b054ad30710e64c0562e
[12] Ishchukova, E., Babenko, L., Anikeev, M. Fast implementation and cryptanalysis of GOST R 34.12-2015 block ciphers // ACM International Conference Proceeding Series. - 2016.
[13] Report on the research work "Development of software and hardware for cryptographic protection of information during its transmission and storage in info communication systems and general-purpose networks" // Committee of Science of the Ministry of Education and Science of the Republic of Kazakhstan, Institute of Information and Computing Technologies, State Registration No. 0118RK01064.