Global Journal of Researches in Engineering: J
General Engineering
Volume 23 Issue 1 Version 1.0 Year 2023
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 2249-4596 & Print ISSN: 0975-5861

New Encryption Algorithm with Improved Security

By Dmitriy Shatokhin

Abstract- The design of new cryptographic algorithms, as a rule, has the main goal of improving their resistance to cryptanalysis methods. Since cryptanalysis methods are constantly being improved, when designing crypto algorithms, it becomes necessary to create new non-standard approaches that can effectively resist the existing cryptanalysis methods. This paper describes in detail a new crypto algorithm created using an original technique aimed at radically improving cryptographic strength. The paper provides both brief theoretical justifications and a complete technical description of the crypto algorithm.

Keywords: cryptography, VOMF technique, stream cipher, cryptographic algorithm, special encryption technique, information security.

GJRE-J Classification: DDC Code: 005.1 LCC Code: QA76.6

New Encryption Algorithm with Improved Security

Dmitriy Shatokhin

Abstract- The design of new cryptographic algorithms, as a rule, has the main goal of improving their resistance to cryptanalysis methods. Since cryptanalysis methods are constantly being improved, when designing crypto algorithms, it becomes necessary to create new non-standard approaches that can effectively resist the existing cryptanalysis methods. This paper describes in detail a new crypto algorithm created using an original technique aimed at radically improving cryptographic strength. The paper provides both brief theoretical justifications and a complete technical description of the crypto algorithm.

Keywords: cryptography, VOMF technique, stream cipher, cryptographic algorithm, special encryption technique, information security.

I. Introduction

One of the fundamental principles of cryptography is that a cryptanalyst's detailed knowledge of a crypto algorithm should not affect the security of the cryptosystem in any way. All existing methods of cryptanalysis are based, one way or another, on a detailed knowledge of the operation of the crypto algorithm under study. It follows that if one somehow limits or reduces the knowledge of the cryptanalyst about the work of at least some important part of the algorithm used, then the cryptanalysis of such a crypto algorithm will turn out to be much more difficult. An example of such an approach is a periodic change in the program code of the encryption function of the crypto algorithm, and such a change should be unpredictable for the cryptanalyst - in particular, it can be based on the encryption key. That is, in other words, the program code of the encryption function of the algorithm (or several functions) is not unchanged and initially defined - instead, during execution it is either replaced by an alternative code from a predetermined large set of functions, or is formed as the algorithm works. On this principle, a special technique for constructing crypto algorithms with increased security is based [1].

This paper describes in detail the synchronous streaming crypto algorithm IMPASE (IMProved Algorithm of Stream Encryption), which is one of the practical examples of the use of a special technique [1], which provides for the presence of a variable program code of the encryption function.

Author: "TechnoCrypt" Research Group, Chief of "TechnoCrypt" Research Group, Karaganda City, Kazakhstan Republic.
e-mails: dvsh68@mail.ru, techno1crypt@gmail.com

II. Abbreviations and Symbols used

KSA: (Key Scheduling Algorithm) is a preliminary procedure for preparing data structures and working variables of the main crypto algorithm based on the encryption key. It precedes the work of the main crypto algorithm.

VOMF: (**VO**latile **M**ain **F**unction) is a special technique for designing crypto algorithms, which provides for the creation of an encryption function (or functions) that changes according to some rules in a crypto algorithm.

IV: Is the initialization vector. A non-secret value sent along with the encrypted message for initialization.

+: arithmetic modulo addition. When adding byte values, the addition is performed modulo 256. When adding 32-bit words, it is performed, respectively, modulo 2^{32}.

*: arithmetic multiplication modulo 2^{32}.

<<<n: cyclic shift of a 32-bit word by n bits to the left.

>>>n: cyclic shift of a 32-bit word by n bits to the right.

<<n: logical shift of a 32-bit word by n bits to the left. The bits that are pushed into the vacated space are always 0.

>>n: is a logical shift of a 32-bit word by n bits to the right. The bits that are pushed into the vacated space are always 0.

&: is a bitwise AND operation.

OR: is a bitwise OR operation.

XOR: is a bitwise XOR operation.

||: is the operation of concatenation (connection) of 4 separate bytes into one 32-bit word.

< >: is the operation of splitting one 32-bit word into 4 separate bytes.

×: is a generic term for some non-linear operation performed on two 32-bit words.

w2b(W): (Word-to-Byte) is a function to convert a 32-bit word *W* to a byte. The conversion is performed by adding together all four bytes of the word *W* modulo 256. The resulting sum of 1 byte is the result of the function.

All illustrative code fragments in this work are written in the standard C language. Hexadecimal numbers are also given in the C format.

III. About the VOMF Technique in the IMPASE Algorithm

As mentioned above, the use of the VOMF technique in the crypto algorithm design is carried out in order to significantly complicate its cryptanalysis. In the general case, for this, the program code of the crypto algorithm provides for one or more changing (volatile) functions that perform data transformation, and which can replace each other in some pseudo-random way. Such volatile functions can either be predefined or created in some way during the execution of the algorithm. In other words, in addition to the uncertainty of key data that is unknown to the cryptanalyst, it is also necessary to increase the degree of uncertainty of the operations performed on this data. There are many practical ways to do this, and the IMPASE crypto algorithm demonstrates just one possible way to use this technique.

a) Volatile Function

The IMPASE crypto algorithm uses a three-argument volatile function with a common prototype as follows:

$$F(A, B, C) = A \times B \times C \qquad (1)$$

where A, B, and C are 32-bit words, and the sign "×" denotes some non-linear operation assigned during execution on 32-bit words. Operations are performed sequentially from left to right, that is, first the operation is performed with arguments A and B, and then – the result of execution with argument C. The order of the arguments can be any, that is, F(A,B,C), F(B,C,A), F(C,B,A), etc. Since there are three arguments, the number of their possible permutations is 3!, that is, 6. The effective order of the arguments when executing the algorithm is determined when executing the key scheduling algorithm (KSA), and is unchanged for the current combination of encryption key and initialization vector.

For this function, 8 special composite non-linear operations have been developed, each of which, as shown in (1), performs a transformation on two 32-bit words, and the result of its execution is one 32-bit word. Before calling the function, 2 operations out of 8 possible are selected in a pseudo-random way. It is easy to show that the number of options for choosing 2 different elements out of 8 possible (taking into account the permutations of these 2 elements) is 56, plus 8 more options in the case of identical elements. Then the total number of options is 64. Given that the result of the function depends both on the order of the operands and on the selected operations, the total number of possible results is 64 * 6 = 384. That is, in other words, 384 variants of the volatile function are possible. All these options somehow depend only on the combination of the encryption key and the initialization vector, and, as was said, the order of the operands is determined during the execution of the KSA, and the operations are selected before the function call, depending on the state of the internal parameters of the algorithm.

An important detail to note is that all these non-linear operations are designed so that they have the same execution time. It is necessary mainly to make run-time cryptanalysis inefficient. This type of cryptanalysis is based on the exact measurement of the execution time of individual sections of the program code of the crypto algorithm. And although in real practical conditions this type of cryptanalysis is very difficult, nevertheless, its possibility must be taken into account.

Creating nonlinear operations with the same execution time is a rather complicated practical problem, and in this algorithm it is solved due to the fact that on most microprocessors, the operations of register cyclic shift, logical register shift, addition, OR and XOR operations are performed in the same number of microprocessor cycles. By combining these operations in a certain way, it is possible to achieve both a good degree of nonlinearity of the result and the same execution time for these composite operations. The following section provides a detailed description of these operations.

b) Nonlinear Operations in the Algorithm

The IMPASE crypto algorithm uses 8 non-linear operations performed in the volatile function described above. The following description of these operations assumes that they are performed on operands A and B. Each operation also uses a 1-byte auxiliary variable *t* to store the result of *w2b* (see Section II), and then truncates *t* to the lowest significant 5 bits.

1. Operation *LrotXor* (Left Rotate and XOR)
```
Usage: LrotXor (A,B)
t=w2b(B) & 0x1F;
LrotXor=(A<<<t) XOR (B>>(32-t));
```

2. Operation *LrotAdd* (Left Rotate and Addition)
```
Usage: LrotAdd(A,B)
t=w2b(B) & 0x1F;
LrotAdd=(A<<<t) + (B<<(32-t));
```

3. Operation *LshOr* (Left Shift and OR)
```
Usage: LshOr(A,B)
t=w2b(B) & 0x1F;
LshOr=(A<<t) OR (B>>(32-t));
```

4. Operation *LshAdd* (Left Shift and Addition)
```
Usage: LshAdd(A,B)
t=w2b(B) & 0x1F;
LshAdd=(A<<(32-t)) + (B<<<t);
```

5. Operation *RrotXor* (Right Rotate and XOR)
```
Usage: RrotXor(A,B)
t=w2b(B) & 0x1F;
RrotXor=(A>>>t) XOR (B<<(32-t));
```

6. Operation *RrotAdd* (Right Rotate and Addition)

```
Usage: RrotAdd(A,B)
t=w2b(B) & 0x1F;
RrotAdd=(A>>>t) XOR (B>>(32-t));
```

7. Operation *RshOr* (Right Shift and OR)

```
Usage: RshOr(A,B)
t=w2b(B) & 0x1F;
RshOr=(A>>t) OR (B<<(32-t));
```

8. Operation *RshAdd* (Right Shift and Addition)

```
Usage: RshAdd(A,B)
t=w2b(B) & 0x1F;
RshAdd=(A>>(32-t)) + (B>>>t);
```

In the IMPASE crypto algorithm, these operations are numbered from 0 to 7. It should be noted that when programming the algorithm, it is advisable to create an array of pointers to functions that perform these operations, and subsequently call these functions by the number of the array element.

IV. Description of the IMPASE Algorithm

Strictly speaking, the IMPASE crypto algorithm is not a separate algorithm - it is a whole family of algorithms, since by changing a number of its parameters, you can get many other similar algorithms with different characteristics. These options are described in more detail in the next section. The following is a description of the basic version of the algorithm.

a) General Characteristics

The IMPASE crypto algorithm is a synchronous stream encryption algorithm that generates one pseudo-random 32-bit word in one cycle of its work. This algorithm is designed primarily for software implementation, but it can also be used in specialized controllers. Being a synchronous stream cipher algorithm, it is actually a generator of a pseudo-random cryptographic sequence (gamma) with a very large period. The sequence obtained as a result of its work can be superimposed on the open data - as a rule, using the "exclusive OR" operation. Thus, the cryptographic strength of such a system is entirely determined by the cryptographic strength of the generated sequence. The following are its main characteristics:

- algorithm type: synchronous stream cipher;
- encryption key size: 384 bits;
- initialization vector size: 32 bits;
- required amount of memory for internal data: about 2.5 kilobytes;
- predefined data and constants: not used;
- output data: crypto-resistant pseudo-random sequence;
- output sequence element: 32-bit word.

Some other properties and features of this algorithm - such as the period of the output pseudo-random sequence, its statistical characteristics, the speed of the algorithm - are described in the next section.

b) Key Scheduling Algorithm (KSA)

This auxiliary algorithm prepares the internal data of the crypto algorithm based on the encryption key and initialization vector. The encryption key, as shown above, has a size of 384 bits and is considered as a sequence of 8-bit blocks, i.e. bytes. All operations with the encryption key, as well as the initialization vector, are performed at the level of an integer number of bytes. Thus, the encryption key must be 48 bytes in size. The initialization vector, respectively, has a size of 4 bytes.

For the operation of the main crypto algorithm, KSA prepares 4 data blocks, which are called S-block (Substitution block), P-block (Parameters block), M-block (Master block) and C-block (Control block). These blocks have the following structure:

- The S-block is 8 byte arrays of 256 bytes each. Each of these arrays contains a permutation of numbers from 0 to 255, thus being a bijection of an ordered array with values from 0 to 255. The permutation in each of the 8 arrays is individual and does not depend on other arrays;
- P-block is a byte array of 259 bytes. During operation, 4 consecutive bytes are extracted from it, interpreted as one 32-bit word;
- M-block is an array of 8 32-bit words;
- C-block is a 4-byte byte array.

When KSA is working, the following operations are strictly sequential:

1. *Formation of the C-block.* To form a C-block (this is an array of 4 bytes), the first 4 bytes of the encryption key are copied into this array.
2. *Formation of the M-block and its copy.* To form an M-block (this is an array of 8 32-bit words), the last 32 bytes of the key are copied into this array, forming 8 32-bit words, since each 32-bit word occupies 4 bytes. Then a copy of the M-block is created - the same array of 8 32-bit words, further referred to as *Mcopy* in the description. It will take part in further KSA operations.
3. *Creation of a shift register with non-linear feedback.* To form the remaining blocks, a byte shift register with non-linear feedback is first created, which will be used as a generator of pseudo-random bytes. The operation scheme of such a register is shown in Fig.1. To do this, the first 16 bytes of the key are copied into a separate 16-byte array *Reg*, and then in this array the first 4 bytes are replaced by 4 bytes of the initialization vector. Further, the array *Reg* is used as a shift register with non-linear feedback. To implement the feedback mechanism, the previously

Global Journal of Researches in Engineering (J) Volume XXIII Issue I Version I Year 2023

created *Mcopy* array of 8 32-bit words is used. A primitive feedback polynomial of the 16th degree as follows is also used:

$$P(X) = X^{16} + X^5 + X^4 + X^3 + 1 \qquad (2)$$

In general, the operation of such a register is almost similar to the operation of classical registers with linear feedback in the Galois configuration [2], with the difference that it uses not a bit array, but a byte array, as well as nonlinear feedback with some features. More details about the operation of shift registers can also be found in [3].

It must be said that the creation and properties of non-linear feedback shift registers is a topic for a separate article, but here it is only important to note that the shift register described above generates a statistically good pseudo-random byte sequence with a very large period.

Initially, after creating this register, 128 "idle" generation cycles are performed - this is necessary for mixing the bits of the key and the initialization vector. Further, this shift register is used as a generator of pseudo-random bytes to create the following blocks.

4. *Formation of the S-block.* To form an S-block, each of the 8 256 byte arrays is first filled with ordered values from 0 to 255. Then the array is shuffled using a pseudo-random sequence obtained from the shift register described above. The following C program fragment illustrates this process. The nextbyte() function is supposed to return the next pseudo-random byte from the register. You also need to remember that in C, arrays are numbered from zero:

```
typedef unsigned char byte;

int i,j;

byte sblock[8][256];

byte t,n;

for(i=0;i<8;i++)

         for(j=0;j<256;j++) sblock[i][j]=j; //filling from 0 to 255

for(i=0;i<8;i++) {

         n=0;

         for(j=0;j<256;j++) { //mixing

             n+=nextbyte(); t=sblock[i][j];

             sblock[i][j]=sblock[i][n]; sblock[i][n]=t;

         }

         for(j=255;j>=0;j--) {

             n+=nextbyte(); t=sblock[i][j];

             sblock[i][j]=sblock[i][n]; sblock[i][n]=t;

         }

}
```

As it follows from this example, shuffling is carried out in two passes - first in the forward direction and then vice versa. After that, the formation of the S-block is over - now all 8 arrays are shuffled independently of each other.

5. *Formation of the P-block.* To form a P-block, the corresponding array of 259 bytes is filled with the next pseudo-random bytes from the register.
6. *Determination of operand indexes for volatile function.* This step determines the order of the operands when the function is called. To do this, three-byte variables are used as indexes - *ix0*, *ix1*

and *ix2*. An array of 3 32-bit words *Opers*, is also created to store the operands of the volatile function. Subsequently, these byte variables will be used in the main cycle of the algorithm as indexes of the *Opers* array. To initialize these variables, the next pseudo-random byte is first generated from the register, and then the following calculations are performed:

```
typedef unsigned char byte;

byte n, ix0, ix1, ix2

n=nextbyte();

ix0=n%3;                                      //ix0 = n mod 3;

if(n>>7) {                                    //if highest bit = 1

  ix1=(ix0+1)%3; ix2=(ix1+1)%3;

  } else {                                    //if highest bit = 0

       ix2=(ix0+1)%3; ix1=(ix2+1)%3;

}
```

As a result, these variables get the values 0, 1, and 2 in some sequence determined by the pseudo-random byte. In this way, the order of the operands is determined. This order is further used in the main cycle of the algorithm.

c) *Main Cycle of the Algorithm*

The main cycle of the IMPASE crypto algorithm consists of two steps. The first step is to extract the next data from the blocks and calculate an intermediate value based on them, and this calculation is performed using a volatile function. The second step is to generate a 32-bit output value and modify the data blocks. During the operation of the main cycle, the algorithm actively uses the operation of concatenation of 4 separate bytes into a 32-bit word, and the reverse operation - the separation of a 32-bit word into 4 separate bytes.

1. *The first step of the main cycle.* The scheme of the first step is shown in Fig.2. The following actions are performed:

– 4 consecutive bytes are extracted from the P-block, starting from the position determined by the first byte of the C-block, i.e. the first byte of the C-block is an index to retrieve the bytes from the P-block. The extracted bytes are concatenated into a 32-bit word and stored as an element of the *Opers[ix2]* array.

– similarly, 4 more consecutive bytes are extracted from the P-block, starting from the position determined by the second byte of the C-block. The extracted bytes are also concatenated into a 32-bit word and stored in the 32-bit *Param1* variable.

– a 32-bit word is extracted from the M-block, while the three highest bits of the third byte of the C-block are used as an index. This word is stored as an element of the *Opers[ix1]* array.

– A 32-bit word is extracted from the S-block as follows: the lowest three bits of the third byte of the C-block are used as the number of the 256-byte array (from 0 to 7). As an index inside the selected array, the fourth byte of the C-block is used with the two lowest bits previously set to zero - this is done to prevent possible overflow of the array. Starting from this index, 4 consecutive bytes are extracted from the array, concatenated into a 32-bit word, then the resulting word is added modulo 2^{32} to the *Param1* variable, and the result is stored as an element of the *Opers[ix0]* array.

– the lowest three bits of the first byte of the C-block are stored in the variable *Op1*. This is the number of the first non-linear operation for a volatile function.

– the lowest three bits of the second byte of the C-block are stored in the variable *Op2*. This is the number of the second non-linear operation for a volatile function.

– a volatile function of the form (1) is called, while the three operands of this function are stored as elements of the *Opers* array, and the numbers of non-linear operations are stored in the variables *Op1* and *Op2*. First, the operation is performed on the elements *Oper[0]* and *Oper[1]*, and then on the result and the element *Oper[2]*. The final result of the calculation is one 32-bit word. It is stored in the *Temp* variable and used further in the second step of the main cycle.

2. *The second step of the main cycle.* The scheme of the second step is shown in Fig.3. The following actions are performed:

- separate bytes are extracted from the S-block as follows. The 32-bit variables *Param1* and *Temp* are separated by 4 bytes each. The lowest three bits of each byte of the *Param1* variable serve as the numbers (0..7) of the 4 arrays in the S-block, and each of the 4 bytes of the *Temp* variable is an index (0..255) for extracting a byte from the corresponding array. For example, if the lowest three bits of each of the 4 bytes of the *Param1* variable are equal to 4,5,0 and 7, and the four bytes of the *Temp* variable are equal to 95,144,67 and 201, respectively, then a byte will be extracted from the 4th S-block array by index 95, from the 5th array - byte at index 144, from the 0th array - byte at index 67, and from the 7th array - byte at index 201. The 4 bytes obtained in this way are concatenated into a 32-bit value, which is the output 32-bit word of the algorithm.
- P-block is modified. To do this, those 4 bytes of the P-block that were used in the first step, and position of which was determined by the first byte of the C-block, are interpreted as a 32-bit word. This word is added modulo 2^{32} to the *Temp* variable (which stores the result of evaluating the volatile function).
- M-block is modified. To do this, the element of the array that was selected at the first step (its index is determined as the value of the three highest bits of the third byte of the C-block) is added modulo 2^{32} to the *Param1* variable.
- C-block is modified. To do this, 4 bytes of the C-block are concatenated into a 32-bit word, which is added modulo 2^{32} to the array element *Opers[ix0]* (this array element, as described above, stores the 32-bit word extracted from the S-block, added to the *Param1* variable).

This completes the main cycle of the algorithm. To generate the next word, the main cycle of the algorithm is repeated. From the description of the main cycle, it can be seen that when generating the output 32-bit word, the algorithm performs a double non-linear transformation: the first is for non-linear operations of calculating an intermediate value, which is performed by a volatile function, and the second is for fetching output bytes from the S-block, which is actually a non-linear replacement operation.

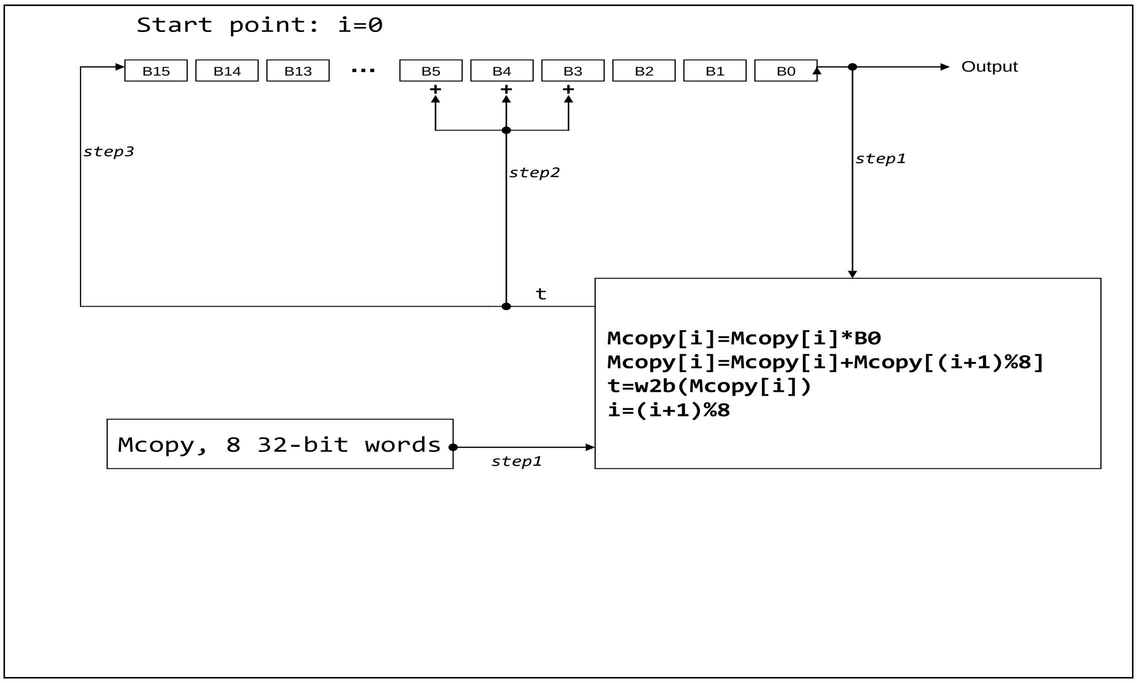

Figure 1: Scheme of Non-linear Feedback Shift Register

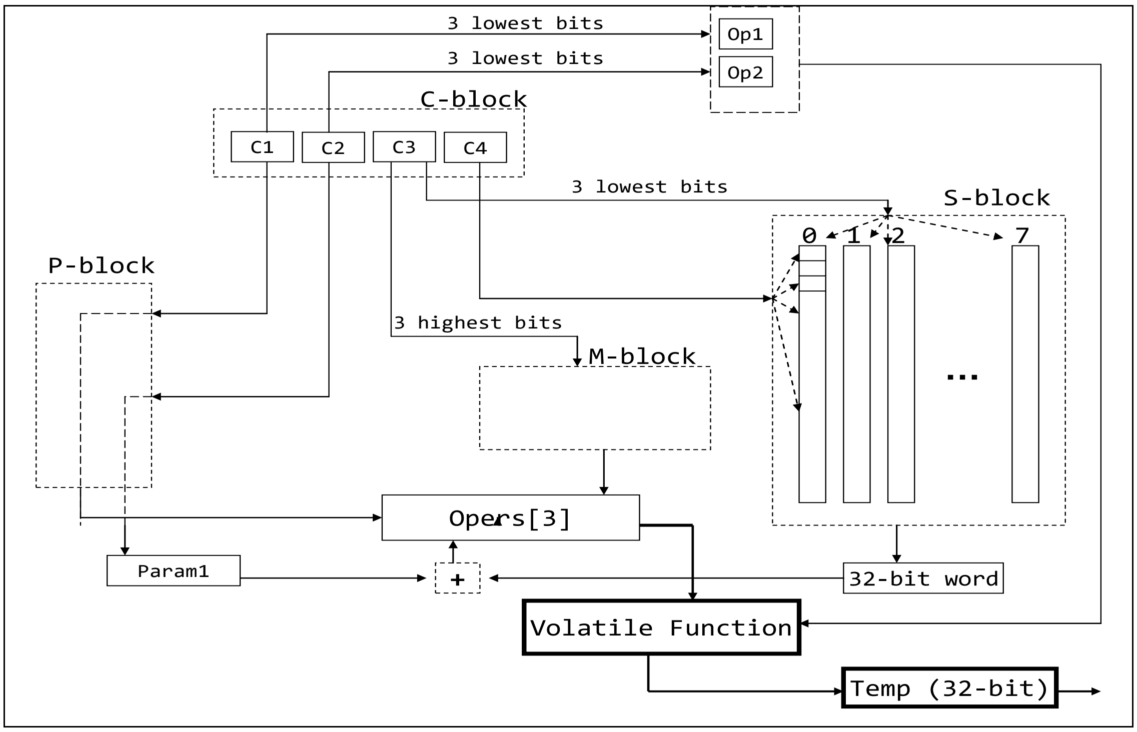

Figure 2: Scheme of the First Step of Main Cycle

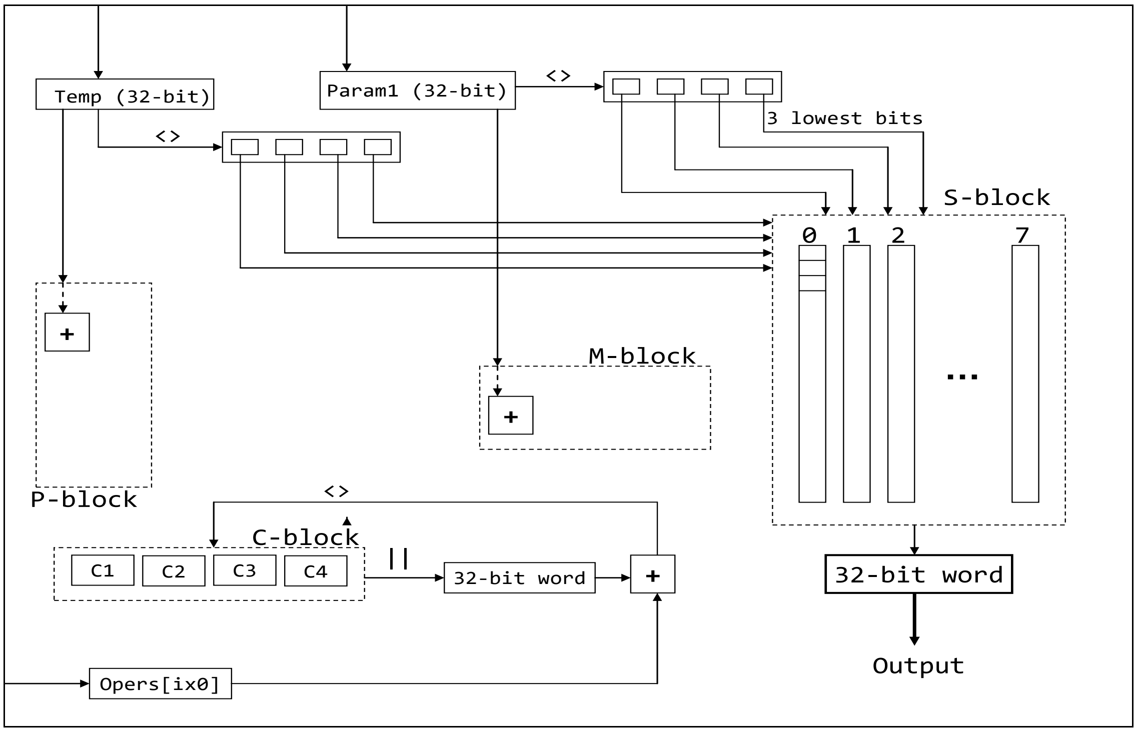

Figure 3: Scheme of the Second Step of Main Cycle

V. Properties of the IMPASE Algorithm

a) Statistical Properties of the Generated Sequence

A necessary condition for the operation of any stream crypto algorithm is, in addition to cryptographic strength, good statistical characteristics of the generated pseudo-random sequence. For the IMPASE algorithm, the statistical analysis of the generated sequence was performed in accordance with the recommended set of statistical tests for pseudorandom sequences [4]. For testing, 16384 32-bit words were generated, which in total is 64 kilobytes.

It should be noted that, as in most crypto algorithms, in IMPASE changing at least one bit of the encryption key or initialization vector leads to a complete

Global Journal of Researches in Engineering (J) Volume XXIII Issue I Version I Year 2023

change in the entire generated sequence, since (as described above) leads to a complete change in the results of the shift register and, accordingly, to a complete change in at least the S-block and P-block. Therefore, statistical tests were carried out for 50 different, completely random, encryption keys and initialization vectors. In all cases, very good test results were obtained.

b) Crypto Algorithm Performance

The pseudo-random sequence generation speed was evaluated on an Intel Core i3 processor with a clock frequency of 3.0 Mhz, on the core of the MS-DOS operating system. This operating system was chosen due to the fact that, due to the absence of third-party processes, the maximum performance of iterative algorithms is achieved on it (unlike Windows and Linux OS). In addition, the MS-DOS kernel is often used in specialized controllers. To evaluate the performance, the algorithm was implemented in the C language, and the Watcom C v9.5a optimizing compiler was used to compile the program. All possible optimizations were made, both at the level of the program code and in the compiler settings. For comparison, we measured the performance of the RC4 algorithms, as well as AES (10 rounds, OFB mode) under the same conditions. The RC4 algorithm was chosen for comparison because it is one of the fastest stream crypto algorithms, and the AES algorithm because it is the current standard. The IMPASE algorithm had a generation rate of 940 Mbps, while the RC4 algorithm had 2200 Mbps, and the AES algorithm had 180 Mbps. It can be assumed that when the algorithm is implemented in the Assembler language, the performance can be significantly improved.

c) Possible Algorithm Modifications

As noted above, the IMPASE algorithm is actually a whole family of algorithms. By changing a number of its parameters, one can obtain crypto algorithms with other properties. The following are the parameters that can significantly affect the properties of the generated sequence:

- the number of arrays in the S-block. Increasing the number of arrays will lead to even more cryptographic strength, but will require more memory to work with, and increase the KSA running time. In the described basic version, the S-block contains 8 arrays and, according to our estimates, this is the minimum required number.
- increasing in the number of non-linear operations for a volatile function. Such an increase will also increase cryptographic strength, since it will lead to an increase in the number of options for a volatile function. However, it must be remembered that the operations must have a high degree of non-linearity, and also have the same execution time - this is the most difficult condition.
- increasing the number of operands in a non-constant function. Such an increase will also increase cryptographic strength, however, it will lead to an increase in the generation time of the output value and, accordingly, to an increase in the generation time of the entire sequence.
- changing the way data blocks are generated. Instead of the existing non-linear feedback shift register, a simpler and faster way to populate and shuffle algorithm data blocks can be designed. This may lead to a simplification of the KSA. However, it must be remembered that the algorithm for preparing data blocks should not contain easily traceable linear dependencies - this can lead to a noticeable drop in the cryptographic strength of the entire algorithm.

VI. Conclusion

This paper describes in detail a method for creating cryptographic algorithms with increased cryptographic strength based on the VOMF technique, and also describes in detail the IMPASE stream crypto algorithm created using this technique, and which is a practical example of its use. It should be noted that the implementation of this technique in the IMPASE algorithm is very simple and intuitive. However, this is only one of the possible ways to apply the VOMF technique - there are many other, more complex ways. In addition, a similar approach can be very successfully implemented also in the design of block crypto algorithms - at present, we are completing tests of a block crypto algorithm built using the same technique.

We hope that this work will be useful for understanding the aspects of improving cryptographic strength in the design and analysis of cryptographic algorithms.

References Références Referencias

1. https://web.archive.org/web/20180411172542id_/https://www.fruct.org/publications/abstract20/files/Sha.pdf.
2. https://en.wikipedia.org/wiki/Linear-feedback_shift_register.
3. B. Schneier, *Applied cryptography.* Second edition. John Wiley & Sons. 1996.
4. Donald E. Knuth, *The Art of Computer Programming*. Vol. 2: Seminumerical Algorithms (3rd ed.). Addison-Wesley Professional. ISBN 978-0-201-89684-8.