MPF based symmetric cipher performance comparison to AES and TDES

A. Mihalkovich[1], M. Levinskas[2], P. Makauskas[3]
Department of Applied Mathematics, Kaunas University of Technology, Kaunas, Lithuania
[1]Corresponding author
E-mail: [1]*aleksejus.michalkovic@ktu.lt*, [2]*matas.levinskas@ktu.edu*, [3]*pijus.makauskas@ktu.edu*

Received 15 March 2022; received in revised form 3 May 2022; accepted 16 May 2022
DOI https://doi.org/10.21595/mme.2022.22517

Abstract. This paper presents a performance comparison of newly created matrix power function (MPF) based symmetric cipher realized in cipher-block chaining mode (CBC) with two other standardized ciphers namely AES-128 and triple DES. The encryption rate is compared using 64 bits arithmetic operations. It is shown that the proposed cipher has a significant advantage against compared ciphers. It is achieved by consuming more memory for MPF realization. Mentioned ciphers are compared using the central processing unit (CPU) the number of clock cycles required to encrypt fixed size plaintexts. This paper also includes a comparison of a number of operations required to encrypt plaintext as well as memory requirements for each of the ciphers with pre-determined parameters. The main difference between symmetric cipher based on MPF and other standardized ciphers mentioned in the paper is the ability to work with larger matrix entries in comparison to other algorithms as well as availability to parallelize the process of encrypting a single block.

Keywords: block ciphers, cryptography, cipher block chaining, encryption, information security.

1. Introduction

The concept of symmetric cryptography has been developed a long time ago. A simple idea of using the same secret key to encrypt a plaintext and to restore it from the received ciphertext was used for centuries. Even nowadays this concept is widely used alongside asymmetric cryptography. Its main advantage over asymmetric encryption is the execution speed i.e., symmetric ciphers are much faster than asymmetric ones. Hence, symmetric ciphers are used to encrypt large amounts of data whereas asymmetric ciphers are used to distribute the secret keys.

In 1949 an American mathematician C. Shannon introduced a concept of perfectly secure ciphers [1]. One of the most intuitive definitions of perfect secrecy states that any ciphertext does not leak any information about the plaintext, i.e. the plaintext μ and its ciphertext c are statistically independent. Mathematically it can be written in the following way:

$$\Pr(c = c_0 | \mu = \mu_0) = \Pr(c = c_0), \tag{1}$$

where Pr $(\cdot)$ denotes the probability of a random event and μ_0, c_0 are fixed. Shannon proved that the one-time pad technique possesses this important property. On the other hand, it has been shown that any cipher with this property uses keys of the size at least equal to the size of the plaintext.

Usually, symmetric cipher uses several rounds to encrypt a plaintext where each round consists of rather simple operations, which make a transformation non-linear. A good example of this approach is the advanced encryption standard (AES) which uses 10, 12, or 14 rounds to encrypt a plaintext [2]. A similar approach was used for the Feistel network presented in 1973 [3]. This method for constructing block ciphers which were used by IBM to develop a data encryption standard (DES) in 1977 [4]. This algorithm was later improved by W. Diffie, M. Hellman, and W. Tuchman who created the triple DES (TDES) algorithm by applying the DES algorithm three times to each block.

In our previous papers we proposed a different approach to the construction of a block cipher

i.e., we suggest trading the multiple rounds technique for the implementation of a single non-linear function called the matrix power function (MPF) to obtain the plaintext. In our paper [6] we proved that our symmetric cipher based on this function possesses the perfect secrecy property and hence does not leak any information about the plaintext [7], [8]. Moreover, in our paper [6] we proposed the cipher block chaining (CBC) mode of the initial Shannon cipher based on MPF. We are making a conjecture that this cipher is based on a one-way function [9]. So far ciphers based on a one-way function that can be used to define an NP-complete problem are conjectured to be resistant to quantum cryptanalysis attacks. Moreover, operations are performed using matrices allowing effective parallelization for our proposed cipher as well as in comparison to the other two algorithms it uses only one round to encrypt the plaintext.

In this paper, we compare the CBC mode of our cipher based on MPF to AES-128 and TDES CBC mode in the sense of clock cycles. Using this method, we can evaluate the execution speed of our algorithm and compare the obtained results to AES-128 and TDES without executing all the schemes on the same operating system. Obviously, this comparison is objective, since the execution time depends only on the total number of clock cycles. Furthermore, we consider the memory requirements of all three CBC modes. We aim to show that our scheme is suitable for implementations in memory restricted environments.

2. Matrix power function and our previous work

The idea of MPF is somewhat similar to regular matrix multiplication and was used in several papers (see e.g. [6], [9]-[11]) by our research group. In general, MPF is a mapping that allows to raise square matrix $\mathbf{Q}$ defined over the multiplicative semigroup $\mathbb{S}$ to the matrix powers $\mathbf{X}$ and $\mathbf{Y}$ defined over the ring of integers $\mathbb{Z}_n$, where $n = ord(\mathbb{S})$ denotes the multiplicative order of $\mathbb{S}$. The notation of this mapping is as follows [9]:

$$\mathbf{X}\mathbf{Q}^{\mathbf{Y}} = \mathbf{E}, \tag{2}$$

where $\mathbf{Q} = \{q_{ij}\}$, $\mathbf{X} = \{x_{ij}\}$, $\mathbf{Y} = \{y_{ij}\}$ and $\mathbf{E} = \{e_{ij}\}$ are square $m \times m$ matrices. Furthermore, $q_{ij}, e_{ij} \in \mathbb{S}$ and $x_{ij}, y_{ij} \in \mathbb{Z}_n$. Every entry of matrix $\mathbf{E}$ can be computed as follows [9]:

$$e_{ij} = \prod_{k=1}^{m}\prod_{l=1}^{m} q_{kl}^{x_{ik}y_{lj}}, \quad i,j = 1,2,\dots,m. \tag{3}$$

As in our previous papers [9]-[12], we refer to matrix $\mathbf{Q}$ as the base matrix. We also refer to matrices $\mathbf{X}$ and $\mathbf{Y}$ as power matrices. Moreover, we call $\mathbb{S}$ a platform semigroup and $\mathbb{Z}_n$ a power ring. We also use notations $Mat_m(\mathbb{S})$ and $Mat_m(\mathbb{Z}_n)$ to identify the sets of $m \times m$ matrices with entries contained in $\mathbb{S}$ and $\mathbb{Z}_n$ respectively.

Previously we have proven several important properties of MPF when the platform semigroup is commutative [6], [9]. In particular, this mapping is associative. Using this fact, the following important property of MPF can be easily derived:

$$\mathbf{X}^{-1}(\mathbf{X}\mathbf{Q}^{\mathbf{Y}})^{\mathbf{Y}^{-1}} = \mathbf{Q}. \tag{4}$$

This property is necessary to restore the initial matrix $\mathbf{Q}$ thus cancelling the effect of MPF. Clearly, this is possible if both matrices $\mathbf{X}$ and $\mathbf{Y}$ are non-singular. We use this property to restore the original plaintext in its matrix form.

Recently we implemented MPF in symmetric cryptography. In our paper [6] we proposed a Shannon block cipher and proved the perfect secrecy property of the encryption function. Empirical results are presented in [12]. There we considered the avalanche effect and the bit independence criterion.

Let us briefly revise the scheme from [6]. Note that the technique we are about to present is a slightly modified version of the encryption scheme proposed in [6]. This was done to eliminate unnecessary additional operations which can be easily avoided.

Prior to key generation step the following public parameters are published [6]:

1) A prime number p and a Sylow group $\mathbb{G}_q$, where q is an odd prime divisor of $p-1$;

2) The ring of integers $\mathbb{Z}_q$;

3) A one-to-one mapping $f: \mathbb{Z}_q \mapsto \mathbb{G}_q$, which is not an isomorphism, i.e. it does not satisfy the following relation:

$$f(x+y) = f(x) * f(y). \tag{5}$$

4) The inverse mapping $f^{-1}: \mathbb{G}_q \mapsto \mathbb{Z}_q$.

We leave the mathematical aspects of generating this data outside of this paper.

Note that the parameter p is important since all the actions within the set $\mathbb{G}_q$ are performed modulo p. Also, actions within the set $\mathbb{Z}_q$ are performed modulo q. In the expressions, we are about to present we suppress these facts to keep them as simple as possible. However, modular arithmetic must always be kept in mind.

The output of the key generation process is the triplet of matrices $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$, where $\mathbf{X}, \mathbf{Y} \in Mat_m(\mathbb{Z}_\mathbf{q})$ and $\mathbf{Z} \in Mat_m(\mathbb{G}_q)$. The matrix $\mathbf{X}$ must not contain zero entries and the power matrix $\mathbf{Y}$ has to be invertible. Gauss-Jordan method could be used to reject singular matrices as it is constructive, i.e. it outputs the inverse matrix $\mathbf{Y}^{-1}$, which may be kept in memory to use during the decryption procedure. There are no constraints on the matrix $\mathbf{Z}$.

Additionally, we define the matrix analogue of the mapping f by its entry-wise application to the argument matrix and denote this matrix mapping by $F: Mat_m(\mathbb{Z}_q) \mapsto Mat(\mathbb{G}_q)$ as well as the inverse matrix mapping $F^{-1}: Mat(\mathbb{G}_q) \mapsto Mat_m(\mathbb{Z}_q)$. Since f does not satisfy the relation Eq. (5), it makes our cipher resistant to algebraic attacks. In other words, the application of function F does not allow to transform the MPF mapping to regular matrix multiplication, thus obtaining a system of linear equations, since $\mathbf{Y}$ is an invertible matrix.

We denote the Hadamard product of matrices by $\odot$ and the inverse of the matrix $\mathbf{A}$ in the Hadamard sense by $\mathbf{A}^H$. Hence, we have:

$$\mathbf{A} \odot \mathbf{A}^H = \mathbf{1}, \tag{6}$$

where $\mathbf{1}$ is a $m \times m$ matrix with all entries equal to the neutral element of the Sylow group $\mathbb{G}_q$.

Let us first assume that the message μ, which needs to be encrypted, consists of a single block. Then it is transformed to a matrix form, which we denote by $\mathbf{M}$. The encryption algorithm is as follows:

$$\mathbf{C}_1 = \mathbf{M} + \mathbf{X},$$
$$\mathbf{C}_2 = \mathbf{Z} \odot {}^{\mathbf{Y}}\mathbf{F}(\mathbf{C}_1)^{\mathbf{Y}},$$
$$\mathbf{C} = F^{-1}(\mathbf{C}_2) + \mathbf{X},$$

where $\mathbf{C}_1$ and $\mathbf{C}_2$ are intermediate results and $\mathbf{C}$ is the matrix representation of the ciphertext. We can now concatenate the entries of this matrix to obtain a bit string c – a final ciphertext of the original plaintext μ.

To decrypt the received ciphertext c this bit string first undergoes a transformation to its matrix form C. Then the following decryption algorithm is executed:

$$\mathbf{D}_1 = \mathbf{C} - \mathbf{X},$$

$$\mathbf{D}_2 = \mathbf{F}(\mathbf{D}_1) \odot \mathbf{Z}^H,$$
$$\mathbf{D} = \mathbf{F}^{-1}\left({}^{\mathbf{Y}^{-1}}\mathbf{D}_2^{\mathbf{Y}^{-1}}\right) - \mathbf{X},$$

where $\mathbf{D}_1$ and $\mathbf{D}_2$ are intermediate results and $\mathbf{D}$ is the matrix representation of the decrypted message.

To shorten this paper, we omit the proof of correctness of this scheme, i.e. demonstrating that $\mathbf{D} = \mathbf{M}$. This proof is based on the algebraic properties of the operations used in both algorithms. Obtaining the original plaintext μ is only a matter of concatenating the entries of the matrix $\mathbf{M}$.

A noticeable feature that distinguishes our cipher from other block ciphers is the fact that it is perfectly secure [6,8], and we can use a single round to obtain a ciphertext. We achieve this important property of our cipher by using several non-linear algebraic operations. However, the proof of perfect secrecy and the security analysis of our scheme is a topic for a separate paper. Here we focus mainly on the performance analysis.

3. CBC mode of our Shannon cipher

The necessity of probabilistic encryption comes from the security consideration of the symmetric ciphers. A widely known fact states that any deterministic cipher no matter how secure is vulnerable to a chosen plaintext attack (CPA). Resistance to this attack is mandatory for any secure symmetric encryption scheme [7]. We are currently working on the proof of this resistance for our scheme.

The probabilistic nature of the obtained ciphertext comes from the introduction of the initialization vector IV, which is chosen at random each time the encryption of the plaintext is performed. Note, that this randomly selected vector is XORed with the segment of the plaintext and the result undergoes the block cipher encryption process. The output of this process is a segment of the final ciphertext and is also used as a new initialization vector for encrypting the next segment of the plaintext.

Based on the discussed general structure of the CBC mode in our paper [6] we presented a probabilistic version of our initial cipher. The major modification we proposed was the introduction of the initialization vector IV to our scheme. Defining the public parameters as in the previous section we encrypt the plaintext μ, which has been divided into n blocks represented by matrices $\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_n$, by executing the following algorithm:

$$\mathbf{C}_{i1} = \mathbf{M}_i + \mathbf{X},$$
$$\mathbf{C}_{i2} = \mathbf{Z} \odot {}^{\mathbf{Y}}\mathbf{F}(\mathbf{C}_{i1})^{\mathbf{Y}},$$
$$\mathbf{C}_i = \mathbf{F}^{-1}(\mathbf{C}_{i2}) + \mathbf{X},$$

where $\mathbf{C}_{i1}$ and $\mathbf{C}_{i2}$ are intermediate results and $\mathbf{C}_i$ is the matrix form of the ciphertext of the block $\mathbf{M}_i$. Moreover, if $i = 1$, then $\mathbf{C}_0 = \mathbf{IV}$ which in our scheme is represented by a randomly selected matrix from the set $Mat(\mathbb{Z}_q)$. The final output of the encryption procedure is a bit string obtained by concatenating the entries of matrices $\mathbf{C}_0, \mathbf{C}_1, \dots, \mathbf{C}_n$ and hence the ciphertext consists of the entries of $n + 1$ blocks. This is a usual experience when applying the CBC mode [7].

The received ciphertext can be decrypted using the same triplet of matrices $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$. This step is performed as follows:

$$\mathbf{D}_{i1} = \mathbf{C}_i - \mathbf{X},$$
$$\mathbf{D}_{i2} = \mathbf{F}(\mathbf{D}_{i1}) \odot \mathbf{Z}^H,$$
$$\mathbf{D} = \mathbf{F}^{-1}\left({}^{\mathbf{Y}^{-1}}\mathbf{D}_{i2}^{\mathbf{Y}^{-1}}\right) - \mathbf{C}_{i-1},$$

where $\mathbf{D}_{i1}$ and $\mathbf{D}_{i2}$ are intermediate results and $\mathbf{D}_i$ is the matrix representation of the i-th block of the decrypted message. To shorten this paper, we omit proving that $\mathbf{D}_i = \mathbf{M}_i$ for all $i = 1,2, \dots, n$.

This fact comes from the algebraic properties of the operations used to construct our technique.

So far standardized algorithms used in CBC mode like AES-128 or TDES do not support parallelization of computations for a single block i.e., there are no options to divide the computations of the encryption algorithm between several processors thus improving the execution time of these schemes. This is not true in our case. Due to matrix actions used in our block cipher multiple options for parallelization of computations exist. As such we can use m or m^2 processors to gain a significant speed boost of the encryption of data.

Keeping this option in mind for now, in the next section we explore the performance of our Shannon cipher in CBC mode and compare the obtained result to AES-128 and TDES used in the same mode.

4. Performance comparison

To perform an objective comparison of three block ciphers we use the notion of a clock cycle – an amount of time between two pulses of an oscillator. To put it simply one clock cycle represents the time during which the smallest unit of processor activity is carried out. For our goals, we assume that all the operations are executed on a 64-bit microprocessor. Hence operations for 64-bit numbers or less would take the same time to execute. However, if we were to consider larger numbers then the execution time of operations increases. For this reason, we limit ourselves to 64-bit numbers and consider the relation between total execution time and matrix order m.

The total amount of clock cycles for our Shannon cipher is calculated by counting the total amount of operations needed to perform each step of the encryption algorithm and evaluating the total amount of clock cycles to perform each individual operation. We rely on the references [13] and [14] for these evaluations.

As such we consider each step of the encryption algorithm of our scheme. The intermediate matrix $\mathbf{C}_{i1}$ is a modular sum of two matrices. If we assume that addition modulo q is performed via the pre-calculated table, then in total it takes $5m^2$ clock cycles to obtain the value of this matrix, since according to [11] table lookup operation takes 5 clock cycles and there are m^2 modular additions in total. Obviously, we have to sacrifice memory to gain the maximum efficiency of this operation.

On the second step of our encryption algorithm, we apply the mapping F, which can also be viewed as a pre-defined table. Then, as in the first step, it takes $5m^2$ clock cycles to apply this transformation to all the entries of a matrix. Since modular multiplication can also be performed via the pre-calculated table, it takes $5m^2$ clock cycles to perform the Hadamard product as well.

The most complex part of the second step is the MPF since it uses multiple modular multiplications and exponentiations. To avoid the computations of modular exponentiations we apply the discrete logarithm mapping hence transforming MPF to a regular product of three matrices. Doing so we get an expression $\mathbf{Y} \cdot ld_g(F(\mathbf{C}_{i1})) \cdot \mathbf{Y}$, where $ld_g(\cdot)$ denotes the discrete logarithm mapping base g – a generator of a Sylow group $\mathbb{G}_q$. Once again applying pre-calculated tables to calculate this expression we see that it takes $20m^3$ to perform two modular matrix multiplications. Additional $10m^2$ clock cycles come from the fact that the discrete logarithm mapping and its inverse were applied to obtain the value of MPF. Hence it takes a total of $20m^3 + 15m^2$ clock cycles to complete the second step and obtain the matrix $\mathbf{C}_{i2}$.

To calculate the ciphertext $\mathbf{C}_i$ we need another $10m^2$ clock cycles to obtain $F^{-1}(\mathbf{C}_{i2})$ and add the matrix $\mathbf{X}$.

Denoting the total amount of clock cycles needed to calculate the ciphertext of a single block by $T(m)$ we get the following expression:

$$T(m) = 25m^3 + 30m^2. \tag{7}$$

As compared to AES our cipher is more flexible i.e., we can freely choose the desired size of

the square matrices m increasing capacity to encrypt larger plaintext taking fewer blocks or to adjust to the length of the ciphertext thus using less junk symbols at the end of the original message to fill the void in the last block. Of course, this is a coin with two sides and hence the choice of public parameters must be supported by the overall security of the cipher and the speed of execution.

By using CBC mode amount of clock cycles does not change, since by construction of our Shannon cipher in both cases we calculate the modular sum between two matrices. It makes no difference if one of the summands is the key matrix $\mathbf{X}$ or the previously obtained matrix $\mathbf{C}_{i-1}$.

We now evaluate the total amount of clock cycles to execute AES-128. According to [13] addition operation, XOR and cyclic shift take 1 clock cycle whereas multiplication takes 2 clock cycles. For simplicity, we assume that all the entries of the matrices mentioned below are 32-bit long. The general structure of a single round of AES cipher is presented below in Fig. 1.

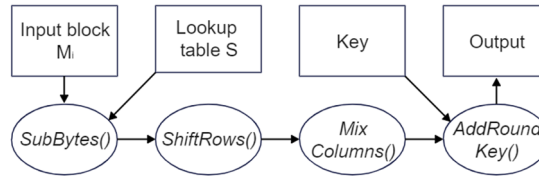


Fig. 1. A general structure of a single round of AES cipher [2]

Note that the presented diagram is valid for all rounds except for the first and the last rounds. In the first round, an extra $AddRoundKey()$ operation is applied at the start of the encryption algorithm. During the last round, the $MixColumns()$ operation is omitted.

Now we evaluate each of these operations individually.

The function $AddRoundKey()$ uses XOR to add two matrices and therefore it takes m^2 clock cycles to obtain the output.

The function $SubBytes()$ is a table lookup operation and hence it takes $5m^2$ clock cycles in total to complete this step for all entries of the initial matrix.

The function $ShifRows()$ does cyclic shifts, which use 1 clock cycle, starting from the second matrix row and therefore it takes $8 \cdot (m^2/2)$ clock cycles in total to complete this step. It treats the row as a 32-bit string which is shifted cyclically by 8, 16, 24 bits, etc. A fixed value of m shifting may be performed in both directions. This can be used to reduce the number of clock cycles.

Finally, the function $MixColumns()$ uses $8m^2$ clock cycles. The idea of these calculations is the same for each entry i.e., it takes 4 XOR operations and 2 multiplications to calculate the result.

Evaluating all the operations performed during a single round we see that all in all, there are:

$$T(m) = 135m^2 \ c.c., \tag{8}$$

for all 10 rounds of encryption. Note that for simplicity we denoted the size of the matrix by m. Based on [2] AES-128 uses a square 4×4 matrix where each entry is Nb bits long and Nb is the block length divided by 32. Hence the total amount of clock cycles needed to encrypt plaintext using AES-128 is $T(4) = 2160$.

Taking into consideration the CBC mode, the main difference is that the XOR operator is applied to the previous ciphertext and the current plaintext which costs additional m^2 cycles. As such we get a total of 2176 clock cycles per block.

We now investigate the total amount of clock cycles to implement the TDES encryption. Since it is essentially the DES algorithm executed three times we focus on this inferior version of encryption.

The general structure of DES is presented below in Fig. 2. In total DES uses 16 rounds to compute the output.

As in the case of AES-128, we consider each operation individually. Even though some of the presented operations are applied to blocks of bits, the same can be done on bit strings thus facilitating our investigation.

The execution of DES starts from permuting the block **M** using the function $IP()$ known as an initial permutation. This is done by shuffling the elements inside the given block according to the pre-defined table **IP** [5]. Hence it is essentially a table lookup operation. Since it transforms 64-bit block preserving its size, it takes 5·64 = 320 clock cycles to complete this operation. We denote the result of this operation by $IP(\mathbf{M}) = \mathbf{IP}$.

After that, the permuted block is split in half using the $Deconcat()$ operation forming two blocks **L** and **R**. Deconcatination takes advantage of two $AND()$ as well as 32 $Shift()$ functions. This stems from the fact, that the number $2^{32} - 1$ is already stored in binary and is used with operation $AND()$ to take only the first 32 bits of the block. After that, the permuted block is shifted by 32 bits, and the $AND()$ operation is repeated. The whole process in detail is performed in the following way:

1) Number $2^{32} - 1$ is extracted from the computer memory and is used together with as the arguments of the $AND()$ operation to get the subblock **R**. This takes 1 clock cycle.

2) After that 32 $Shift()$ operations are applied to $IP(\mathbf{M})$ and the resulting block together with the number $2^{32} - 1$ are used as the arguments of the $AND()$ operation. All in all, these actions take 34 clock cycles in total, since 32 clock cycles are needed to perform shifts and 2 clock cycles are needed to perform $AND()$ operation. The outcome of these actions is the subblock **L**.

It is important to note at this point, that DES has one KEY bundle from which sixteen 48-bit long keys are generated using key schedule protocol. Each key $\mathbf{K_j}, j = \overline{1,16}$ is used in one of the 16 rounds of the DES cipher as it is somewhat iterative. In each iteration, the function f is applied to the block **R**. The result is XORed with the block **L**.

Note that the function f is the most time-consuming part of the DES cipher. This function uses a total of 10 pre-determined tables. Hence for the most part this function consists of table lookups which take 5 clock cycles each. In Fig. 2 we present the structure of this function and later explore its performance.

We can see that the function f starts out by using the permutation table **E** to map a 32-bit block **R** to a 48-bit block which takes 5·48 = 240 clock cycles. This block is then

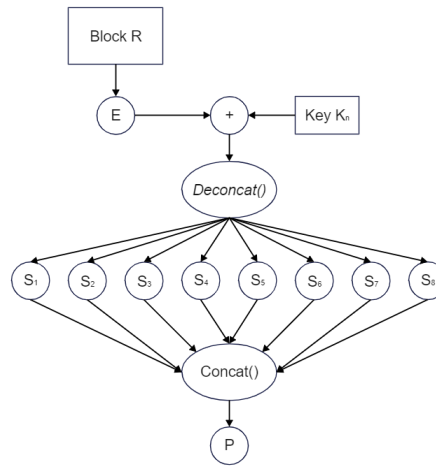


Fig. 2. A general structure the DES cipher function f [5]

XORed with $\mathbf{K}_j$ taking 241 clock cycles in total. The resulting block is then deconcatinated into 8 strings of bits of the same 6-bit length. This deconcatination follows from an analogous approach as the one performed before, taking $7(8-1)+1 = 50$ clock cycles in total.

Each one of the resulting strings undergoes one of the functions $S_k(\,)$, $k = \overline{1,8}$. These functions transform a 6-bit vector into a 4-bit one by performing lookups in the appropriate tables. The lookups are carried out by using the first and last bits of the argument string to form a row number and the remaining middle bits to form a column number. It takes 4 clock cycles in total to extract the least and most significant bits of the string whereas their concatenation requires 2 clock cycles to carry out a single leftward shift and a XOR operation. The column number is formed 2 in clock cycles using a single shift and a XOR. Finally, a 4-bit value is selected from the appropriate table $S_k(\,)$. Overall, eight functions take $8(6 + 2 + 5) = 104$ clock cycles to produce eight 4-bit strings which we denote by $\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_8$.

The obtained strings are then concatenated firstly by taking $\mathbf{s}_1$ and multiplying by 2^4 (this takes 2 clock cycles) in order to shift the bits leftwards and make way for $\mathbf{s}_3$. These values are XORed using 1 clock cycle. The result is multiplied by to make way for and the XOR operation is applied again. This procedure continues until all the strigs are glued together. In total it takes $3(8 - 1) = 21$ clock cycles.

The last part of the function f is a simple bijective permutation P that uses yet another table to map the concatenated block of 32 bits to a permuted block of the same length. This takes $5 \cdot 32 = 160$ clock cycles in total.

Given the description of all the parts that define f, we see, that the amount of clock cycles required for the function f to be executed is 576.

The output of the function f is the XORed with **L** using 1 clock cycle. Keeping in mind that this happens 16 times we get a total of 9232 clock cycles.

Lastly, the subblocks **L** and **R** are concatenated together by retrieving and multiplying **L** by 2^{32} in 2 clock cycles to make way for the bits of **R**. As previously XOR operation is used to add **R**, all in all, taking 3 clock cycles. It remains to use the final permutation which in fact is the inverse of the initial permutation. This takes 5·64 = 320 clock cycles as we are dealing with table lookups once again.

Having evaluated all the operations described above, it is easy to see that the total amount of clock cycles to execute DES encryption is 9909. Hence for TDES, we get:

$$T = 3 \cdot 9909 = 29727\ c.c. \tag{9}$$

Now that we explored all three block ciphers and the CBC modes of these ciphers, we can compare their performance to encrypt a plaintext. Since for our cipher we can vary the value of matrix order m whereas this parameter is fixed for AES and DES, we focus on 4×4 matrices for MPF based CBC mode. We motivate this choice by the fact that AES-128 has the same value of m fixed, although other values can be considered as well using expression (9). Obviously, having more blocks instead of having higher matrix dimensions would give better results in terms of clock cycles. Be aware, however, that by enlarging matrices dimensions the idea of CBC mode becomes irrelevant since based on plaintext size it could be encrypted using 1 block. This negatively affects the key size i.e., it grows rapidly.

Another important fact to note is that the block **M** which represents a segment of the plaintext in the matrix form is an element of the set $Mat_4()$ and hence each entry of this matrix can be encoded by a maximum of 63 bits. However, for practical purposes, we consider values $p = 4079$ and $q = 2039$ in Table 1 as we did in our previous paper [6]. This comparison was chosen because the difference between taken plaintexts is the smallest and all entries for all algorithms are taken into account calculating clock cycles.

Additional columns were added allowing to determine how many blocks are required to encrypt chosen size plaintext.

Based on the values in Table 1 we can see that if we consider AES-128 with smaller plaintext e.g. 384 bytes, the number of blocks would go down to 24 blocks, but MPF based CBC mode should still require 18 blocks with junk at the end. However, we can see that MPF based CBC

mode still can encrypt this plaintext faster based on the evaluated clock cycles. As for the DES algorithm - it takes a message in binary, splits it into square blocks of 64 binary entries, and applies a series of various transformations on each block. Obviously, the same observation regarding the message size. stays true for TDES as well.

Table 1. Clock cycles comparison

Plain text size (bytes)	MPF based CBC mode		AES-128 CBC mode		TDES CBC mode	
	No. of blocks	Clock cycles	No. of blocks	Clock cycles	No. of blocks	Clock cycles
512 (506)	23	40480	32	69632	64	1902528
2048 (2024)	92	161920	128	278528	256	7610112
8192 (8096)	368	647680	512	1114112	1024	30440448

5. Size and memory comparison

Besides knowing the clock cycles needed to encrypt the data, it is very important to know how much memory space the whole process could take. In the previous section, we mentioned that modular arithmetic operations can be treated as table lookups. This section will involve calculations of memory resources as well as additional memory requirements for key storage.

MPF based CBC mode uses as a key a triplet of $m \times m$ matrices $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$. Entries of matrices $\mathbf{X}$ and $\mathbf{Y}$ are encoded by a $\lceil \log_2 q \rceil$ bits whereas entries of the matrix $\mathbf{Z}$ are encoded by $\lceil \log_2 p \rceil$ bits. Considering $m \times m$ matrices, it would take:

$$2m^2 \cdot \lceil \log_2 q \rceil + m^2 \cdot \lceil \log_2 p \rceil$$

bits of memory. Aside from these three matrices, it also uses tables for modular arithmetic, discrete logarithm mapping and its inverse, mapping f and its inverse. Modular arithmetic tables have the size $q \times q$, where each entry is encoded by a $\lceil \log_2 q \rceil$ bits in case of addition and bits for multiplication. Tables for discrete logarithm and the mapping f together with their inverses are much more concise since they can be represented by a single vector of q entries encoded by $\lceil \log_2 p \rceil$ bits each. Finally, the overall memory requirements for MPF based CBC mode is the following function of q:

$$2\lceil \log_2 q \rceil (q^2 + 2q + m^2) + m^2 \cdot \lceil \log_2 p \rceil.$$

For our chosen values ($p = 4079$, $q = 2039$, $m = 4$) we need roughly 12 megabytes of memory space. Smaller parameters would result in faster encryption and less memory size, but based on [12], it would not be as secure to ensure the randomness of the entries in the obtained ciphertext matrix. All in all, parameters should be chosen based on the device memory, processing properties, and the size of the text messages.

AES-128 algorithm uses 128 bits key and a fixed table from where the values are taken. The mentioned matrix is a 16×16 size table with 8-bit entries in total giving 2048 bits. All other encryption procedures use XOR operation or shifts which does not require addition table creation. Therefore, taking into consideration the size of the key and table with predefined values it would take 2176 bits of memory.

In the DES case, there is a 64-bit key bundle KEY consisting of sixteen 48-bit keys generated using the key schedule. This adds up to be 832 bits stored. There are also several permutation tables in DES, which in total use 192 bits of memory as well as some solitary values for operations that use up to 96 bits altogether. This, coupled with the fact that every $\mathbf{S}_k$ table uses 128 bits of memory, takes 3808 bits for TDES in total.

6. Conclusions

In this paper, we considered a CBC mode of a perfectly secure Shannon cipher based on the MPF. The main parameters of our cipher are primes p and q, where q is a divisor of $p-1$, and the order of matrices m. Relying on observations of this paper these parameters have to be chosen keeping in mind the encryption rate, available memory resources, and the length of the message.

To speed up our algorithm we settled for storing several tables in the memory of the device. This experience is nothing new since the TDES algorithm uses a similar approach. However, we can see from Section 5 that this solution has its drawbacks as well. Obviously, modular arithmetic tables take most of the space as compared with other tables mentioned in that section. Evaluating this fact, we do not recommend to use large values of p and q, but rather settle for values of q encoded by roughly 16 bits, since otherwise, the memory requirements may outweigh the speed of our scheme. Based on this fact, we used the values $p = 4079$ and $q = 2039$ for comparison thus keeping our algorithm secure, fast, and suitable for implementation in memory restricted environments.

The comparison was done using the notion of the clock cycle. This makes it objective since the operations used for the construction of three considered ciphers differ and hence, we need a single unit to measure the encryption rate. We think that our choice of this unit is reasonable since it allows us to compare basic versions of all three ciphers without worrying about speed boosts available for AES and TDES. The results of this comparison have shown that for the chosen values of the main parameters our scheme is faster than the two other ciphers we considered. Specifically, our cipher is roughly 1.5 times faster than AES-128 and roughly 47 times faster than TDES.

Moreover, for our cipher, it is possible to alter parameters p and m which would allow adjusting the speed and memory requirements of the algorithm depending on the available resources. For AES and TDES algorithms the values of considered parameters and look-up tables are recommended by the National Institute of Security and Technologies.

However, one significant advantage, which was not explored in this paper is the parallelization possibility. This allows us to use multiple processors to boost the speed of our algorithm to encrypt a single block. This is something that distinguishes our schemes from AES and DES as these algorithms do not support this feature. Furthermore, this ability may be useful for exploring larger values of the matrix order m since the total amount of clock cycles to encrypt a single block is a cubic polynomial of m as can be seen in Eq. (7). Future research may be done on exploiting this possibility.

References

[1] C. E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, Vol. 28, No. 4, pp. 656–715, Oct. 1949, https://doi.org/10.1002/j.1538-7305.1949.tb00928.x

[2] J. M. Dworkin et al., "Advanced encryption standard (AES)," National Institute of Standards and Technology, Gaithersburg, MD, Advanced Encryption Standard, Nov. 2001.

[3] H. Feistel, "Cryptography and Computer Privacy," *Scientific American*, Vol. 228, No. 5, pp. 15–23, 1973.

[4] "Data Encryption Standard (DES)," Federal Information Processing Standards Publication 197, United States National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 1977.

[5] E. Barker and N. Mouha, "Recommendation for the Triple Data Encryption Algorithm (TDEA) block cipher," National Institute of Standards and Technology, Gaithersburg, MD, Special Publication (NIST SP), Nov. 2017.

[6] E. Sakalauskas, L. Dindienė, A. Kilčiauskas, and K. Lukšys, "Perfectly secure Shannon cipher construction based on the matrix power function," *Symmetry*, Vol. 12, No. 5, p. 860, May 2020, https://doi.org/10.3390/sym12050860

[7] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC Press, 2020, https://doi.org/10.1201/9781351133036

[8] D. Boneh and V. Shoup. "A Graduate Course in Applied Cryptography," https://toc.cryptobook.us/

[9] E. Sakalauskas and K. Luksys, "Matrix power function and its application to block cipher s-box construction," *International Journal of Innovative Computing, Information and Control*, Vol. 8, No. 4, pp. 2655–2664, 2012.

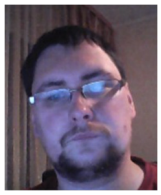
[10] A. Mihalkovič and E. Sakalauskas, "Asymmetric cipher based on MPF and its security parameters evaluation," *Lietuvos Matematikos Rinkinys*, Vol. 53, pp. 72–77, Dec. 2012, https://doi.org/10.15388/lmr.a.2012.13

[11] A. Mihalkovich, E. Sakalauskas, and A. Venckauskas, "New asymmetric cipher based on matrix power function and its implementation in microprocessors efficiency investigation," *Electronics and Electrical Engineering*, Vol. 19, No. 10, pp. 119–122, Dec. 2013, https://doi.org/10.5755/j01.eee.19.10.5906

[12] A. Mihalkovich and M. Levinskas, "Avalanche effect and bit independence criterion of perfectly secure Shannon cipher based on matrix power," *Mathematical Models in Engineering*, Vol. 7, No. 3, pp. 50–53, Sep. 2021, https://doi.org/10.21595/mme.2021.22234

[13] A. Fog. "Instruction tables," https://www.agner.org/optimize/instruction_tables.pdf

[14] S. Goldshtein and D. Zurbalev, *Pro.NET Performance – Optimize Your C# Application*. 2012.



Aleksejus Mihalkovich received Ph.D. degree in Kaunas University of Technology, in 2015. Since 2019 he is an Assist. Professor in the Department of Applied Mathematics and a member of Cryptography and Blockchain Technology research group. His current research interests include the cryptanalysis of symmetric and asymmetric cryptographic primitives.

Matas Levinskas pursuing master's degree of Applied Mathematics in Kaunas University of Technology. A member of Cryptography and Blockchain Technology research group.

Pijus Makauskas pursuing master's degree of Applied Mathematics in Kaunas University of Technology. A member of Cryptography and Blockchain Technology research group.