Transactions on Engineering and Computing Sciences - Vol. 11, No. 6
Publication Date: December 25, 2023
DOI:10.14738/tecs.116.15857.
Noureldien, N. A., & Hassan, M. B. (2023). NMC: A Fast and Secure ARX Cipher. Transactions on Engineering and Computing Sciences. 11(6). 45-59.
SERVICES FOR SCIENCE AND EDUCATION

NMC: A Fast and Secure ARX Cipher

Noureldien A. Noureldien
University of Science and Technology,
Faculty of Computer Science and Information Technology, Omdurman, Sudan

Mustafa B. Hassan
University of Science and Technology,
Faculty of Computer Science and Information Technology, Omdurman, Sudan

ABSTRACT

Fastening complex computing tasks and reducing latency have become essential in areas such as cryptography, robotics, natural-language processing, and video streaming. This is because even a few milliseconds can have a significant impact on the operation, performance, and computational cycle consumption of the algorithm. This paper introduces a new ARX cipher design called NMC. NMC is a fast and secure stream cipher and it generates keystream bytes solely from secret key bits and plaintext message length. NMC has high randomness, long keystream period, and one time pad (OTP) keystream production. Experimental testing confirmed the randomness of the keystreams and showed that NMC has marginally better speed performance than RC4 for short messages but is slightly slower for messages of average length and slightly large messages. NMC's synchronization mode showed that the keystream period is long and NMC achieved OTP keystream production. The results indicate that NMC is a competitive and promising cipher algorithm.

Keywords: Symmetric Encryption, Stream Cipher, Keystream, Randomness, Keystream Period, Cipher Performance, Throughput.

INTRODUCTION

The two basic cryptographic approaches are the symmetric and asymmetric key cryptography methods. In the first approach, the sender and receiver use the same secret key to encrypt and decrypt a plaintext message. In the second approach, each sender and receiver have two keys: public and private. The sender encrypts the plaintext message using the public key of the receiver, whereas the receiver decrypts the ciphertext message using his private key. There are several variants of these approaches.

Symmetric-key cryptography algorithms are either block or stream ciphers. Block ciphers divide plaintext into blocks of fixed size and encrypt or decrypt the data block as a whole. Stream ciphers consider plaintext as a continuous stream of bits or bytes. Typically, a stream cipher uses a secret key and public initial vector (IV) value to generate a sequence of bits or bytes called the keystream. The keystream is then used to encrypt/decrypt plaintext using an XOR operation [1].

Stream ciphers are based on the Vernam cipher and exploit the concept of a one-time pad (OTP) [2]. OTP (One-Time Pad) uses a completely random keystream that is unbreakable by attackers, whereas stream ciphers use a pseudo-random keystream [3]. Therefore, in the design of a stream cipher, the keystream must be sufficiently random to render it resistant to attacks that aim to expose the secret key. The execution of modern stream ciphers usually consists of two stages: initialization and keystream generation. In the initialization stage, the stream cipher first loads the secret key and the initialization vector (IV) into an internal state. It then updates the internal state by iteratively calling an updating function for a sufficient number of rounds to thoroughly mix the secret key and IV. The objective of mixing is to increase confusion, which helps hide the relationship between the ciphertext and the secret key. Therefore, stream ciphers rely solely on the confusion. In the keystream generation stage, the stream cipher generates keystream bits or bytes from a highly confused internal state by invoking an output function, while continuously updating the internal state.

Stream ciphers are classified as either synchronous or self-synchronized. In a synchronous stream cipher, the keystream is produced independently of the plaintext and the current internal state is determined solely by the previous internal state [4]. However, if a character in a plaintext message is changed, only a single character in the ciphertext is affected. This means that the error does not spread to the rest of the ciphertext messages [5]. On the other hand, in a self-synchronizing cipher, the keystream is generated based on previous plaintext or ciphertext, with the goal of making security analysis more difficult [6].

The security and speed of the cipher are two important aspects [7]. Security refers to the cipher's ability to withstand different attacks, whereas speed refers to the quick execution of encryption and decryption processes. Both aspects are determined by the design of the cipher. In cipher implementation, programmers can utilize various optimization techniques to maximize their speed [8].

The design of stream ciphers using linear feedback shift register (LFSRs) is widely used [9]. An LFSR is a simple device that generates a long sequence of bits before it begins repeating itself. Stream ciphers typically combine multiple LFSRs to increase the period of the generated bit sequence, which creates the keystream. However, stream ciphers based solely on LFSRs are vulnerable to various types of attack [10].

Another popular design is the ARX design. In this design, the ciphers only use modular addition, rotation, and XOR (ARX) operations. ARX cipher design is known to make ciphers fast, and their implementation is efficient. Examples of ARX ciphers include the popular RC4, Salsa20 family of stream ciphers, ChaCha, HC-128, and Speck family of lightweight block ciphers [11, 12, 13, 14]. In contrast to the simple operations of ARX ciphers, large tables and tree-party machines have been proposed [15].

Several schemes have been proposed to enhance the security of the primary LFSR design by incorporating nonlinearity. Examples of these schemes include clock-control generators, combined functions, and filter generators. Nonlinear feedback shift registers (NFSRs) are another technique used to introduce nonlinearity [16, 17]

Many international standards use stream cipher algorithms to protect confidential data by encrypting and decrypting them. Examples of such ciphers include the use of the RC4 stream cipher in the wireless network standard IEEE 802.11 [18], SNOW 3G and ZUC stream ciphers for the 3GPP standard [19, 20], and the E0 stream cipher for Bluetooth systems [21].

Developing faster algorithms is an important issue in many areas; however, it remains an elusive goal for data scientists. The ability to accelerate complex computing tasks and reduce latency is essential in areas such as cryptography, robotics, natural-language processing, video streaming, and gaming. Milliseconds can have a significant impact on the algorithm's operation, performance, and number of computing cycles consumed [22].

In this study, we introduced a new ARX stream cipher called NMC. The design of NMC generates a keystream using ARX operations, the secret key, and the length of the plaintext message. The generated keystreams passed NIST randomness tests, and the NMC throughput was satisfactory.

CIPHER SPECIFICATIONS

In this section, we describe the stream cipher NMC. The cipher produced the stream cipher from a 256-bit secret key K and a 32-bit subkey S. The length of the keystream is the same as the length of plaintext/ciphertext message M.

Operations, Variables, and Functions

The following operations are used in NMC design

- $\oplus$ bit-wise exclusive OR
- || concatenation
- % Modular addition
- >>> right rotation operator
- [x, n] the position of a bit: [x, n] means the n^{th} bit of the binary string x
- | x | the length of: | x | = n means the binary string x is of length n bits
- B[x] the binary form: B[x]=str means the binary string equivalent to the decimal value x is str.
- D[str] the decimal form: D[str]= x means the decimal equivalent of the binary string str is x.
- R[B] read block: R[B] means read block of bits from the EK.

The following variables are used:

- **M**: the length of the message,
- **K**: the secret key binary string,
- **B**: block of bits
- **EK**: Extended Key,
- **S**: the Subkey.
- **Lengthofblock**: the length of the block to be read from the secret key,
- **Block-start**: the bit from which block starts.
- **Block-end**: the bit at which the block ends.

- **KS**: a string of keystream bytes.

There are two functions being used in the cipher:

- $f_1(B_i)$ = this function determines the length of the block B_i, i.e., $|B_i|$
- $f_2(x, y, z,)$: This function is used to updates the variables (Lengthofblock, block-start, block-end).

Initialization Process

The initialization process of the cipher consists of: expanding the key K from 256-bit to 1024-bit, initialization of the subkey S, the initialization of the parameters needed to read the first block from producing of the secret key (Lengthofblock, block-start, block-end).

The Expanding of the Secret Key K:

- Let K = K0||K1||K2||K3 ||K4||K5||K6||K7||, where each Ki denotes a 32-bit value
- The key K is expanded to 1024-bits as follows:

  for (i=8 to 11); (i=12 to 15); (i=16 to 19)

  $K_i = k_j \oplus K_l$, where j+l=7; j+l=11; j+l=15 respectively; and

  for (i=20 to 23)); (i=24 to 27); (i=28 to 31)

  $K_i = k_j \oplus K_l$, where j+l=23; j+l=27; j+l=31 respectively;

Thus, the expanded Key is:

EK= K0||K1||K2||K3 ||K4||K5||K6||K7||…….. ||K30||K31||

i.e EK= $b_0, b_1, b_2, \ldots\ldots\ldots\ldots\ldots\ldots, b_{1022}, b_{1023}$

The Subkey S is Initializes as follows:

1. Let | S | =0
2. Let | M | = m;
3. Let m'=m;
4. For i=0 to 31

   If [K, m] > |K|

   >>> K

   else

   S = S|| [K, m]

   m = m+m'

The Initialization of the Parameters of the First Block

1. Let | M | = m
2. Let B[m] = str
3. if |str| > 8

   Lengthofblock=|str|

   else

   Lengthofblock=8
4. Block-start=0
5. Block-end=Lengthofblock

The Keystream Generation Algorithm

For a plaintext/ciphertext message of length M bytes, a keystream of length M bytes is generated. The generation of each keystream byte is produced by an iteration of the algorithm. Keystream generation also makes use of the $f_2(x, y, z,)$ function to update the internal state variables after each iteration. During each iteration one block B_i is extracted from EK. These B_i's will be of variable lengths. The length of the block is determined by the function $f_1(B_i)$. **Figure** (1) shows the algorithm, and Figure (2) shows a general diagram for the algorithm

Input: {EK, M}//Expanded Key (EK) and length of the message. (M)

1. dec_0= D [R[B_0]]
2. dec_s = D[S]
3. KS_0= B [($dec_0 \oplus des_s$) %256]
4. i = 1
5. while (i < M) {
6. if (Lengthofblock > |EK|) {
7. EK >>>1}
8. else
9. {
10. $f_1(B_i)$
11. aa=D [R[Bi]]
12. dec_i= aa $\oplus$ dec_{i-1}
13. KS_i= B [($dec_i \oplus des_s$) %256]
14. KS = KS || KS_i
15. $f_2(x, y, z,)$
16. }
17. i++;
18. }

Output: KS = KS_0||KS_1||KS_2||……|| KS_{M-1}; //Keystream of length M byte

Figure (10): Keystream Production algorithm

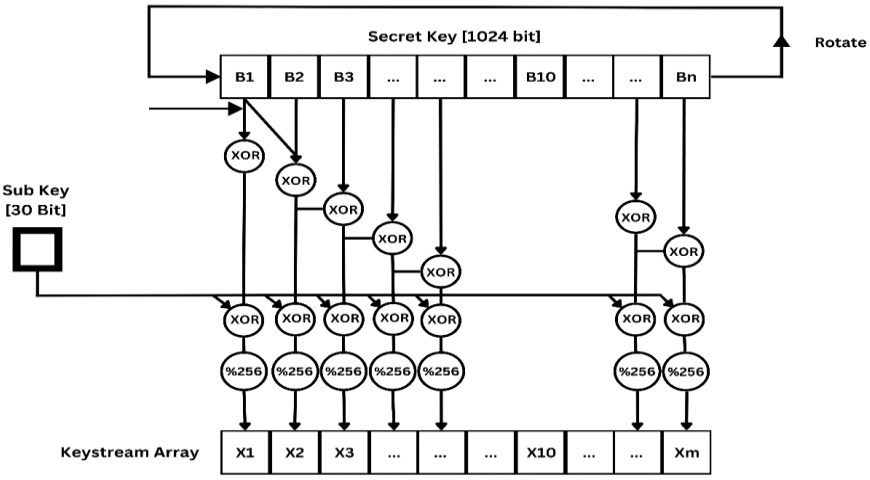


Figure (2): Keystream Production

NMC SECURITY FEATURES

In this section, we discuss some of the security features of the NMC. This discussion reveals the strengths and weaknesses of the cipher. A good stream cipher must maintain three important design considerations [23].

A cipher encryption sequence should have a long period.

This means that the pseudo-random function used to produce the keystream bits/bytes will eventually repeat. Thus, the longer the period of repeating the keystream bits and bytes, the more difficult it is to perform cryptanalysis to reveal the secret key.

The keystream should approximate the properties of a true random number stream as closely as

This means that the produced keystream must possess the properties of truly random sequences. The more randomization there is in the keystream, the more randomized the ciphertext, which makes cryptanalysis more difficult.

The cipher must resist brute-force attacks.

This means that to protect against brute-force attacks, the key must be sufficiently long or the cipher must adopt one-time pad (OTP) encryption. Based on the above three considerations, we discuss the NMC features.

NMC Operation Modes

NMC operates in two modes: discrete and synchronization. In discrete mode, the initial state= Init (State) is retained after each encryption or decryption process, whereas in synchronization mode, the system preserves its current state, and the next encryption or decryption process starts by using the current state as its initial state.

Discrete Mode:

In this mode, all the variables and constants that define the initial state=Init (State) (see Section 2.1) return to their initial state values before the beginning of the next encryption or decryption process. This means that if we re-encrypt the same message, then the same keystream will be reproduced. If the second plaintext message to be encrypted is different from the first message, then the produced ciphertext will be different and unique regardless of whether it has the same length as the first message or not.

Synchronization Mode:

In this mode, if we look at the internal state after the encryption/decryption process is completed and immediately before the next cryptographic process begins, we find that the variables EK, S, Block-start, Block-end, Lengthofblock, preserve their current states. This means that if we re-encrypt the same message, the produced key stream will be different from the previous one. This mode must synchronize the NMC system on the sender and receiver sides to allow the system to retain its internal state after each encryption or decryption operation.

The Keystream Period of NMC

In the NMC synchronization mode, the transformation function= F(state) that updates the initial state while producing keystream bytes is defined by the operations of the key stream production algorithm. If we assume that all secret key bits can be read by B blocks before a rotation is made and the length of the plaintext is M, such that M > B*1024, then at least 1024 rotations will be made. If we look at the internal state after 1024 rotations, we find that the only variable that returns to its initial state is the variable EK, and all remaining state variables maintain their current state values. This implies that retaining the full E(state) is unpredictable. This ensured that the keystream period was extremely large. However, the exact period of NMC was difficult to calculate.

Randomness of the Keystream

When designing a stream cipher, the keystream must be sufficiently random to make it difficult for attackers to reveal the secret key**.**

To check the degree of randomness of the produced keystream against a truly random sequence, we have to carry out what are known as "randomness tests." This is typically performed using a test statistic. A test statistic is designed to follow a certain probability distribution *(e.g., P)* for a truly random sequence. When a test statistic is applied to the binary sequence to be tested, the value *v* is computed using the statistical test, and the probability of v under the probability distribution *P* is computed. This computed probability is known as the "*p*-value." If the *p*-value is greater than or equal to a certain threshold, the binary sequence is considered truly random. Otherwise, it is not [24].

Many statistical test suites are available today to perform randomness tests, such as NIST [25], DIEHARD [26], and the Federal Information Processing Standard (FIPS) [27]. The National Institute of Standards and Technology (NIST) has developed 15 randomness tests to test the randomness of binary sequences. Every statistical test defined in the NIST test suite takes the binary sequence as input and outputs a p-value that determines whether the binary sequence is random according to the property specified by the test.

To test the randomness of the keystream produced by a cipher, a *p*-value was determined for each test in the NIST test suite. If $p = 1$, the given keystream is said to be a perfectly random sequence, and if $p = 0$, the given keystream is said to be a non-random sequence. The significance level for the tests was set at 0*:*01. If $p \geq \alpha = 0{:}01$, then, with **a** confidence level of 99%, we can say that the produced keystream is random, and if $p < \alpha = 0{:}01$, then the keystream is said to be non-random with a confidence level of 99%.

To test the randomness of the keystreams produced by the NMC/1024-bit, we generated five pseudo-random keys of length 1024 bits, and we used these keys to produce keystreams of length 1048576 (220) bits. The NIST test suite was applied to each produced keystream. The hexadecimal values of the keys are listed below, and Table (1) shows the results of the NIST tests.

To test the randomness of the shorter keystreams, we used Key1 to produce seven keystreams with lengths of 524288, 65536, 32768, 1024, 512, 256, and 64 bits. Table (2) shows the results of the NIST tests.

Key1:
HEX:
A1C4BCE65BA8E10526999FD24653462547B008FBBCBF3B2FEBA21B31AA98B5B10316699D33AC11ACA258CAABD5EF3B5D172939C7351A92929FA8D8718B23C5A82A2C145FA92786D8B8C2AD3B1A0847703FE0DEB13371C8F2DDB97CE87553B6E3E2FAF535881D37B5C3ED2366C1A3B0545F012742D193A47DA1C4415FC7CB85B3.

Key2:
HEX:
771E0930277735238DB655F3BFEA4572D2F49074CB28D3DDF76700648A37CA3D7D8431F0183B3B4CB0B8232F0169A003A4B3F176E6E75B1F9B2B5313FA4D2930921F44C2BDF62C9F74AF9F00EA97DE886F41896E0A9B18CFA393AC8D328E92D3CDB74B4D17BB472A7C9141A5F41A87AE1E206AED1FB6C690B3FE0088D5EE325D

Key3:
HEX:
F195DA14E884804826EA5F0B79DFF5B05D09FC39185EAD6DAFC174E43B8F2DE3F6D5E0E28FD026BC6472640A1776474B635115732D01988CCEAC14CBDFD1D50A42BC87319F207266F96D5301A9F4009D5FC6C266D6F72A36C258AF7821421E450055C804340D521D8A1755AB0C467C0FB778AD63F8123B05EE6209D0BE4C479B

Key4:
HEX:
224F1E250D89C88B6277EBBDBE211CF2DE4BBFD637F017165663386C528EB772B27E54C2D53125C349EDFC00E66BA9A9C6401ED2DB06A409C706F46809E433A9AFEFE74E41E1A97CEC18C4345C3E98EE65F8BD2C7CAD7CBA2357AB0764D17576DA10FEE5E9EFAD5CCB18732EBB7FF34A51F657BA605F0AD1174C0FB06F324AF0

Key5:
HEX:
30F03D4C544014ECF66E4EBEEBCA8CF3E6A62B9FC3E021C534A406A772C49BA48EBEE418B59C42BC9EE61A9288FE7E37845CC8BC7903237F32827A865A9E3E76F1C11D8306B95D827A61A5771CDB421F66669128E2AAE26844F9CDA19E9546EF1E3520D02C232E5672FCB1E1711C3EECF6EBBC4714C7BF7148A3A7344BFA7BC9

Table (1): NIST Randomness Test for Keystream of length 1048576-bit (2^{20} bit)

No	Test	Key1	Key2	Key3	Key4	Key5
1	Frequency	0.843045	0.934647	0.122105	0.039208	0.706917
2	Frequency Test within a Block	0.999819	0.342376	3.422544	0.999999	0.978323
3	Runs	0.810299	0.931461	0.337865	0.026811	0.778963

4	The Longest Run of Ones in a Block	0.049540	0.536554	0.401555	2.641118	0.089606
5	Binary Matrix Rank	0.238095	0.451316	0.746515	0.617303	0.421067
6	Non-overlapping Template Matching	0.331140	0.797788	0.888071	0.284408	0.216905
7	Overlapping Template Matching	0.392010	0.022561	0.102453	0.233181	0.473590
8	Maurer's "Universal Statistical"	0.487205	0.494424	0.727239	0.323932	0.032540
9	Linear Complexity	0.461928	0.475290	0.597584	0.957495	0.026613
10	Serial Test	1: 0.952751 2: 0.810330	1: 0.993133 2: 0.933056	1: 0.191661 2: 0.339072	1: 0.010424 2: 0.027244	1: 0.451357 2: 0.228589
11	Approximate Entropy	0.998088	0.860160	0.402997	0.051617	0.167188
12	Cumulative Sums (Cusum)	Forward: 0.996260 Reverse: 1	Forward: 0.983174 Reverse: 0.998482	Forward: 0.055472 Reverse: 1	Forward: 0.075788 Reverse: 0.503696	0.986001 Reverse: 0.988732
13	Random Excursions	0.118282	0.026618	0.316593	0.270731	0.397902
14	Random Excursions Variant	0.161318	0.012203	0.200351	0.089701	0.234913

Table (2): NIST Randomness Test for different Keystream lengths using Key1

	Key Length = 1024 Bit						
Test	P-Value						
	Keystream 524288 bits	Keystream 65536 bits	Keystream 32768 bits	Keystream 1024 bits	Keystream 512 bits	Keystream 256 bits	Keystream 64 bits
Frequency	0.881431	0.552679	0.573110	0.531971	0.859684	0.900523	0.900524
Frequency Test within a Block	0.991758	0.728543	0.863041	0.723713	0.468438	0.526963	0.526963
Runs	0.991160	0.904569	0.996546	0.230075	0.156869	0.706917	0.706917
Longest Run of Ones in Block	0.325267	0.060707	0.589664	0.515257	0.701481	0.312395	0.312395
Approximate Entropy	0.675670	0.971916	0.986276	0.652262	0.219538	0.211142	0.211142
Cumulative Sums	Forward: 0.998668 Forward: 0.998668	Forward: 0.712842 Reverse: 1	Forward: 0.753033 Reverse: 0.670167	Forward: 0.774936 Forward: 0.774936	Forward: 0.856882 Reverse: 1	Forward: 0.906386 Reverse: 0.987102	Forward: 0.906386 Reverse: 0.987102

From Tables (1) and (2), we observe that the five keystreams passed all tests except three out of 70 tests, and the seven short keystreams passed all tests except one out of 42 tests. Because each of the failed tests occurs only once, we can interpret such failures as statistical anomalies.

Adopting a True One-Time Pad (OPT)

In stream ciphers, a "one-time pad" is a stream cipher in which a randomly generated keystream is used only once to encrypt the message. Thus, for NMC to implement the OTP, the keystream produced by the algorithm must be a random keystream and unique for the given encryption session. This means that the keystream cannot be reproduced, even when the same plain text is re-encrypted.

In synchronization mode, NMC can implement a true OTP, which means that even if the same plaintext message is re-encrypted, NMC will produce two different keystreams and thus two different ciphertexts. This becomes possible because blocks read out of the secret key in the re-encryption process will be different from those read in the first encryption. This is because reading blocks start where the last block of the previous encryption ends. The other two NMC operations, the secret key rotation and XORing of blocks, support the uniqueness of the produced keystream. This means that NMC produces any keystream, regardless of its length, only once, thereby achieving OTP keystream production.

NMC Performance

To measure the performance of NMC, we used throughput, which is calculated as the total encrypted plaintext (in bytes) divided by the encryption time (in milliseconds). To evaluate the NMC throughput experimentally, we compared its performance with that of RC4 with a 1024-bit key by conducting three experiments.

The experimental results are listed in Tables (3-7). Experiments were performed using a laptop with an Intel® Core™ i5-2410M CPU, 2.30 GHz, 8 GB of installed memory (RAM), and a 64-bit Windows 7 operating system. The codes were written in Java 10, and no optimization techniques were applied. The data used in all tests were stored in text files that had to be read by the algorithm codes.

In the first experiment, we measured the throughput for short messages with 10, 20, 40, 60, 80, 100, 200, 300, 400, and 500 bytes. Each row in Table (3) shows the two best readings out of the five rounds. Each round consisted of 10 iterations (i.e., the best two readings out of the 50 tests). Table (3) and Table (6) show the throughput, and Figure (3) shows a chart of the speeds of NMC and RC4 for short messages.

In the second experiment, we measured the throughput of slightly larger messages of 1, 2, 5, 10, 50, and 100 KB. Each row in Table (4) shows the two best readings out of five. Table (6) shows the throughput, and Figure (4) shows a chart of the speed of NMC and RC4 for average-length messages.

In the third experiment, we measured the throughput for average length messages of 1 MB, 2 MB, 5 MB, 10 MB, 15 MB, and 20 MB. Each row in Table (5) shows the two best readings out of the three rounds. Each round consisted of 10 iterations (i.e., the best two readings out of the 30 tests). Tables (5) and (6) show the throughput, and Figure (6) shows a chart of the speed of NMC and RC4 for slightly large messages.

From Table (3), it is clear that the NMC throughput is similar to that of RC4 for short messages of up to 500 bytes. For the average length messages in Table (4) and slightly large messages in Table (5), NMC achieves 60% and 86% of RC4 throughput, respectively. Figure (10) shows the comparative throughput results, which demonstrate that NMC is a promising algorithm.

Table (3): Speed of NMC and RC4 for Short Messages.

Text File Size	RC4			NMC		
	Time1/ms	Time2/ms	Average/ms	Time1/ms	Time2/ms	Average/ms
10 bytes	0.160	0.164	0.162	0.136	0.154	0.145
20 bytes	0.131	0.134	0.133	0.161	0.161	0.161
40 bytes	0.141	0.145	0.143	0.111	0.117	0.114
60 bytes	0.161	0.161	0.161	0.120	0.123	0.122
80 bytes	0.175	0.179	0.177	0.140	0.145	0.143
100 bytes	0.168	0.176	0.172	0.162	0.167	0.165
200 bytes	0.266	0.267	0.267	0.191	0.197	0.194
300 bytes	0.342	0.342	0.342	0.253	0.262	0.258
400 bytes	0.218	0.224	0.221	0.316	0.317	0.317
500 bytes	0.282	0.341	0.312	0.385	0.386	0.386
Total=1710			Sum =2.090			Sum= 2.005

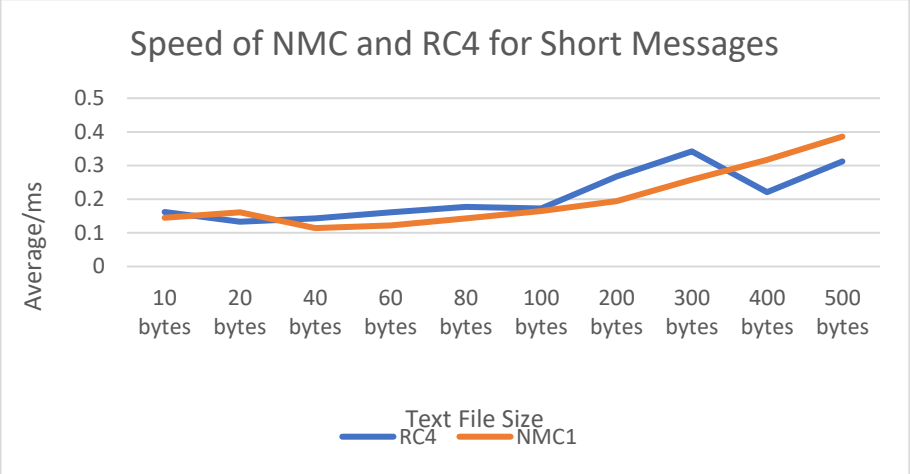


Figure (3): Speed of NMC and RC4 for Short Messages.

Table (4): Speed of NMC and RC4 for Average Length Messages

	RC4			NMC		
Text File Size	Time1/ms	Time2/ms	Average/ms	Time1/ms	Time 2/ms	Average/ms
1 KB	0.502	0.508	0.505	0.448	0.452	0.450
2 KB	0.511	0.526	0.517	0.633	0.638	0.636
5KB	0.975	0.987	0.981	1.430	1.470	1.145
10 KB	1.63	1.67	1.650	3.220	3.280	3.250
50KB	7.43	7.44	7.440	13.65	13.690	13.670
100 KB	13.92	13.98	13.950	24.10	24.900	24.500
Total=172032			Sum= 25.043			Sum= 43.651

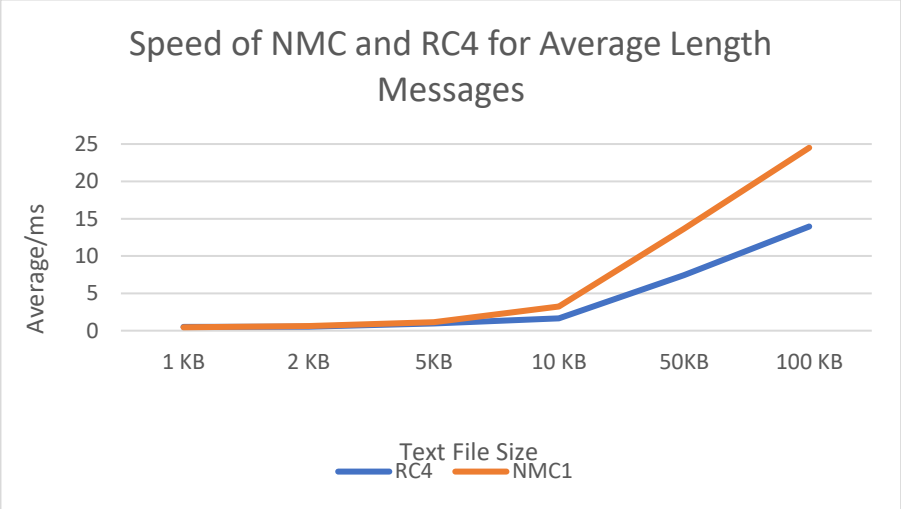


Figure (4): Speed of NMC and RC4 for Average Length Messages.

Table (5): Speed of NMC and RC4 for Slightly Large Messages

	RC4			NMC		
Text File Size	Time1/ms	Time2/ms	Average/ms	Time1/ms	Time 2/ms	Average/ms
1 MB	138	139	138.5	200	210	205
2 MB	276	276	276	361	367	364
5 MB	693	693	693	839	840	839.5
10 MB	1389	1391	1390	1600	1611	1600.5
15 MB	2075	2081	2078	2379	2391	2385
20 MB	2790	2791	2790.5	3150	3155	3152.5
Total= 53 MB			Sum=7366			Sum= 8546

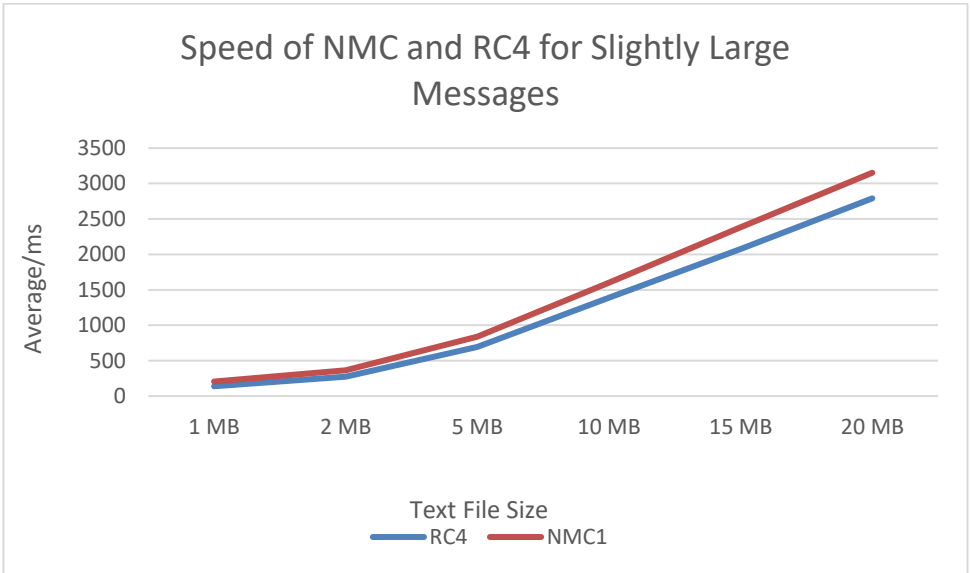


Figure (5): Speed of NMC and RC4 for Slightly Large Messages.

Table (6): NMC and RC4 Throughput

	Slight Large Messages	Average Length Messages	Short Messages
RC4	7.4 MB/second	6.9 MB/second	819KB/second
NMC	6.4 MB/second	3.9 MB/second	852KB/second

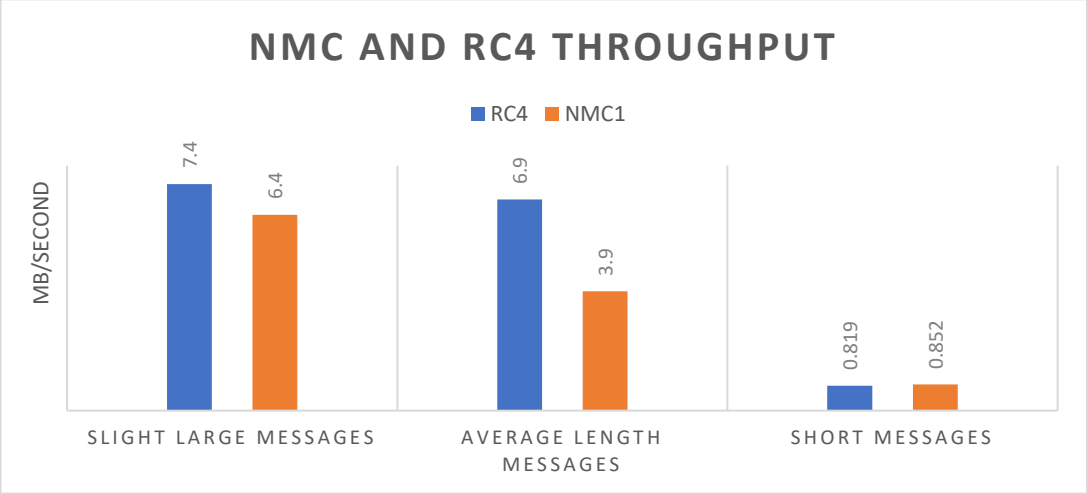


Figure (6): NMC and RC4 Throughput

CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

This study introduces a new cipher that retains the One-Time Pad security feature and has promising performance, especially for short messages. This makes it suitable for applications that use short messages. The following points highlight our upcoming work on improving the MNC1.

- Security analysis of NMC must be performed.
- Therefore, the speed of the algorithm must be improved.
- The correlation between the keystream and key must be investigated.
- The diffusion test to satisfy the diffusion property—each bit of the key should affect the keystream with equal probability—has to be performed.
- Additional randomness tests in the keystream must be performed out.

References

[1] A. Klein, Introduction to stream ciphers, Stream Ciphers, Springer, (2013), pp. 1-13.

[2] S. M. Bellovin, Frank Miller: Inventor of the One Time Pad, Cryptologia, vol. 35, issue 3, (2011), pp. 203-222.

[3] C. E. Shannon, Communication theory of secrecy systems, Bell system technical journal, vol. 28, issue 4, (1949), pp. 656-715.

[4] C. Fontaine, Synchronous stream cipher. In: Encyclopedia of Cryptography and Security. Beilin: Springer, 2005. 1274–1275

[5] W. Millan, E. Dawson, On the security of self-synchronous ciphers. In: Proceedings of Australasian Conference on Information Security and Privacy, 1997. 159–170

[6] C. Manifavas, G. Hatzivasilis, K. Fysarakis, I. Papaefstathiou, A survey of lightweight stream ciphers for embedded systems, Security and Communication Networks, 2010.

[7] Lin JIAO, Yonglin HAO & Dengguo FENG, Stream cipher designs: a review, SCIENCE CHINA, Information Sciences, March 2020, Vol. 63 131101:1–131101:25

[8] Kuai Yu, et.al. "Efficient Software Implementation of ZUC Stream Cipher", Proceedings of the 2nd International Conference on Vision, Image and Signal ProcessingAugust 2018, Article No.: 52, pp 1–6https://doi.org/10.1145/3271553.3271580

[9] A. Makalesi, et al., "A New pseudo random number generator design with LFSR based 32-bit floating point with high-speed FPGA," Int. J. Adv. Eng. Pure Sci., 32(3), 219-228 (2020)

[10] B. Qu, D. Gu, Z. Guo, J. Liu, "Differential power analysis of stream ciphers with LFSRs," Computers & Mathematics with Applications, 65(9), 1291-1299 (2013).

[11] G. Paul, S. Maitra, RC4 stream cipher and its variants, Discrete Mathematics and Its Applications, CRC press, (2011).

[12] D. J. Bernstein. The Salsa20 family of stream ciphers, CR.YP.TO, (2007), available online: http://cr.yp.to/snuffle/salsafamily-20071225.pdf, last visit: 01.07.2015.

[13] B. Zhang, Z. Shi, C. Xu, Y. Yao, Z. Li, Sablier v1, CAESAR competition, (2014), available online: http://competitions.cr.yp.to/round1/sablierv1.pdf, last visit: 01.07.2015.

[14] Beaulieu, Ray; Shors, Douglas; Smith, Jason; Treatman-Clark, Stefan; Weeks, Bryan; Wingers, Louis. "The SIMON and SPECK Families of Lightweight Block Ciphers", 2013.

[15] M. Rosen-Zvi, E. K. Ido Kanter, W. Kinzel, Mutual learning in a tree parity machine and its application to cryptography, Physical Review E, vol. 66, issue 6, (2002), pp. 066135-066148.

[16] T. Chen, L. Ge, X. Wang, C. Jiamei, TinyStream: A Lightweight and Novel Stream Cipher Scheme for Wireless Sensor Networks, CIS 2010, (2010), pp. 528- 532.

[17] M. Hell, T. Johansson, W. Meier, Grain a Stream Cipher for Constrained Environments, International Journal of Wireless and Mobile Computing, vol. 2, No 1/2007, 2007, pp. 86-93.

[18] Rivest R. The RC4 encryption algorithm. Rsa Data Secur Inc Doc No, 1992, 20: 86–96

[19] ETSI/SAGE. Specification of the 3GPP confidentiality and integrity algorithms UEA2&UIA2. Document 2: SNOW 3G Specification, Version 1.1, 2006. http://www.gsmworld.com/using/algorithms/docs/etsi sage 06 09 06.pdf

[20] Feng X T. ZUC algorithm: 3GPP LTE international encryption standard. China Inform Secur, 2011, 19: 45–46

[21] Bluetooth. Specification of the Bluetooth system. 2005. https://www.bluetooth.com/specifications/adoptedspecifications

[22] Samuel Greengard, Better Algorithms through Faster Math, June 2023 | Vol. 66 | No. 6 | Communications of the ACM.

[23] Kumar, I. Cryptology. Laguna Hills, CA: Aegean Park Press, 1997

[24] Nagendar Yerukala, V Kamakshi Prasad, and Allam Apparao, Performances and Statistical Analysis of Stream Ciphers in GSM Communications, Journal of Communications Software and Systems, Vol. 16, No. 1. March 2020.

[25] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. A statistical test suite for random and pseudorandom number generators for cryptographic applications, National Institute of Standards and Technology (NIST), special publication 800-22.

[26] G. Marsaglia. (1995), DIEHARD Statistical Tests. http://stat.fsu.edu/geo/diehard.html

[27] NIST. (2001), Federal Information Processing Standards (FIPS) Publications: FIPS 140-2, Security Requirements for Cryptographic Modules, http://csrc.nist.gov/publications/PubsFIPS.html#140-2.