Journal of Information Hiding and Multimedia Signal Processing ©2022 ISSN 2073-4212
Ubiquitous International Volume **13**, Number **1**, March **2022**

A Novel Approach to Improve the Performance of Serpent Algorithm using Lorenz 96 Chaos-based Block Key Generation

Huwaida T. Elshoush

Computer Science Department
Faculty of Mathematical Sciences and Informatics
University of Khartoum, Sudan
htelshoush@uofk.edu

Khalil T. Obeid

Computer Science Department
Faculty of Mathematical Sciences and Informatics
University of Khartoum, Sudan
khalil.obeid@outlook.com

Mahmoud M. Mahmoud

Computer Science Department
Faculty of Mathematical Sciences and Informatics
University of Khartoum, Sudan
Mahmoud.mustafa101@gmail.com

Received September 2021; revised December 2021

Abstract. *This paper presents a novel approach to enhance the security and performance of Serpent algorithm. A sub key is generated for each block using block key generation (BKG) algorithm that is based on Lorenz 96 chaos and then the processes of encryption and decryption are run in parallel. The proposed approach is applied and tested using forkjoinPool framework on java platform. The results were compared with Serpent algorithm running sequentially and with another Serpent algorithm using block key generation. The approach achieved better execution time than both of them, furthermore with high reduction rate in encryption execution time for large number of blocks. While the reduction was not significant for small number of blocks, for large number of blocks the implementation results showed reduction in time of up to 53.2%. Furthermore, the reduction rate is compared with prevailing methods achieving great results. Thus, the approach demonstrated the potential of the improved Serpent algorithm with Lorenz 96 choas-based BKG and gave favorable results.*

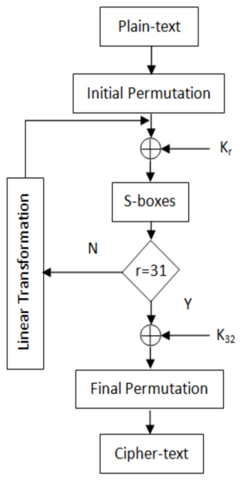
Keywords: Serpent, Lorenz 96, Chaotic map, Block Key Generation, Parallel Computing

1. Introduction.

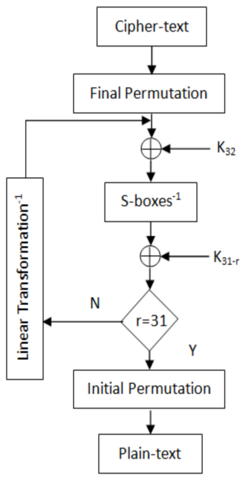
Sending and receiving data is a key element of computer network. The type of data exchanged differs in secrecy. Data can be classified as secret such as in personal information, and confidential or private in military and banking transactions. One of the most important requirements of these networks is to provide secure transmission of information. Cryptography is one of the techniques to provide the secure way to transfer the important information [1].

A replacement to Data Encryption Standard (DES) was needed by the US National Institute of Standards and Technology [2] in 1997. Advance Encryption Standard (AES) and Serpent were the top candidates. Eventually, AES was chosen because it is faster than Serpent although the later was more secure [2][3]. Furthermore, memory requirement and execution time were impediment to its choice. The 32 rounds of Serpent influence the performance directly.

Serpent was developed by Ross Anderson (University of Cambridge Computer Laboratory), Eli Biham (Technion Israeli Institute of Technology), and Lars Knudsen (University of Bergen Norway) [4]. Being is a symmetric block cipher, it uses 256 bit key to encrypt 128 bit of plaintext. Figure 1 depicts the details of the Serpent block cipher. It uses three main functions as elucidated herein:



(a) The encryption process.



(b) The decryption process.

FIGURE 1. The Serpent Algorithm [5]

1. **Initial Permutation (IP)** The initial permutation of bits is performed using a lookup permutation table to determine which bit to put in which position. Equally, it could be done algorithmically by substituting a bit at index i with the bit at index $i * 32 \bmod 127$, where only bits 0 and 127 are kept in their positions [6].
2. **Round function (R)**
   The round function is performed 32 times. Each round consist of three operations: key mixing, s-boxes substitution, and linear transformation; except in the last round, the linear transformation is replaced by an additional key mixing operation as shown in figure 2 [6].
   - **Serpent Round Keys Generation**
     For the 32 Serpent rounds, 33 round keys should be created from the user key. In the first place, eight 32 bit keys, namely w_1 to w_8 will be generated by dividing the user key into 32 bits. Then, 132 intermediate keys are created using the next pseudo code:

```
For $i = 8$ to $131$
        $w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus -w_{i-1} \oplus phi \oplus i) <<< 11$
```

     Note that:
     $\oplus$ is the exclusive-or operation
     phi is known as golden ratio (hexadecimal $0x9e3779b9$)
     $<<<$ is a left rotation

Thereafter, running the intermediate keys through the S-boxes, the 33 round keys are hence created and then amalgamated into 128-bit blocks, as demonstrated in figure 2.

3. **Final Permutation (FP)** The inverse of the initial permutation is the final permutation. It is carried out to place the bits back into the right positions. It can be performed through lookup table or algorithmically by substituting the bit at position i with bit at position *i*4 mod 127*, where only bits *0* and *127* are in their actual places. This produces the final output cipher text [6].

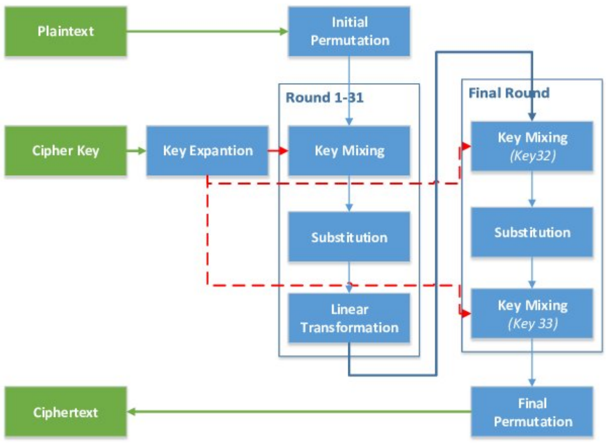


FIGURE 2. Serpent block encryption diagram [7]

The decryption is similar to encryption process but with the inverse ordering of keys.

Consequently, the initiation of this work is precisely to improve the performance and security of Serpent. Ergo, the proposed approach improves the execution time of Serpent by using parallel computing to speed-up encryption and decryption processes [26]. Parallel computation is a method in which several computations can be done simultaneously on two or more microprocessors. It can also be performed using multicore and multiprocessor computers [8]. Furthermore, the proposed approach enhances the security by generating sub keys for each block using two approaches; namely block key generation cipher mode, and Lorenz 96 choas-based block key generation algorithms [26]. The execution performance of the two generated keys approaches are hence compared.

The remainder of the paper is structured as follows: the recent techniques in improving the performance and security of the Serpent are discussed in the next section. Section 3 the two proposed key generation methods whilst the proposed method is elucidated in section 4. The experimental results and analysis are represented in section 5. Finally, section 6 concludes the paper.

2. **Related work.**

Recently, many techniques to improve the performance and security of Serpent were introduced by many researchers. A brief survey of the proposed modifications and their analysis are explained in this section.

In March 2000, Osvik et all [9] proposed hardware improvements to enhance Serpent by speeding up the algorithm in 32-bit processors by reducing the number of registers needed using boolean operations to be suitable for 32-bit processors that have 8 registers. Moreover, two way parallel execution is utilized. Although their results show enhancement, the authors recommended trying three way parallel execution for finer results. Researchers [10] attempted also to improve the Serpent using Hardware.

In July 2013, Fouda et al [11] proposed image block encryption algorithm based on logistic map and linear chaotic map: PWLCM. The proposed function used permutation and diffusion keys that were generated using chaotic system and Linear Diophantine Equation (LDE). The proposed algorithm was evaluated using speed analysis, entropy analysis, key space analysis, differential attack analysis, key sensitivity analysis, and other analysis. The results show that the proposed method can generate large permutation and diffusion keys very fast also have faster and higher security level.

A year later, in October 2014, Taher et all [12] suggested another hardware implementation for Serpent called Field Program-mable Gated Arrays (FPGA) device. Their work provide reliability and hardware-time speed, thus Serpent runs faster.

In December 2015, Ali et all [13] use Serpent in encrypting image by splitting it into 512-bit blocks and divide each block into 4 128-bit blocks. Afterwards, encrypt the last block using Serpent and expand it into 3 blocks, then next XOR the new blocks with the other blocks. This produces faster encryption and decryption processes.

In January 2016, Singh et al [14] designed an algorithm that generated multiple keys from single user defined key called Block Key Generation (BKG). Thus every block has its own key and does not depend on the output of the other blocks. This can make every block runs independently and removes any data pattern. This function runs like EBC with key generation function. It takes the user key and the previous block key as input and returns new key in four steps. Initially, it runs the previous block key (or the user key if it runs on the first block) in a permutation function. The output of this function is run on exclusive OR function with the user key. Then the output of the exclusive OR function is processed on hash function SHA-256 that returns 256 bit key. The last step trims the key into 128 bit key and uses it as a key to the AES function.

A month later, Pendli et al [1] proposed to use parallel computing to enhance the execution time of AES algorithm on multiple cores processors. They used Open which is an API for parallelization based on fork and join technique. The number of blocks to be encrypted or decrypted is divided by 2 and each n/2 block is processed on a different core. The implementation results showed reduction in time. This reduction started with 38% for small files size and the percentage increased up to 45% for large files. At the end, they concluded that it is possible to use the multi-core processors for parallel implementation of AES algorithm and similar algorithms.

In July 2016, Altigani et al [15] tested the performance of the National Institute for Standards and Technology (NIST) AES operation modes: ECB, CBC, CFB, OFB and CTR using textual data as 100 characters for small data, 1000 characters for medium data, and 10000 characters for large data. The results set CFB as best mode with small input data, but with large and medium data CTR was the best one. On the other hand, ECB was the worst mode when using small and medium data with huge difference than others, but in large data it was the second best mode.

In August 2017, S.M.H.Alwahbani and H.T.Elshoush [16] proposed a chaos based audio steganography and cryptography method. It is a higher Least Significant Bit (LSB) layers algorithm in which the secret message is encrypted first by one-time pad algorithm, then hid using steganography. Two chaotic sequences of Piecewise Linear Chaotic Map (PWLCM) were used. In the encryption process, the key for one-time pad is generated by PWLCM chaotic map. In the steganography process, the second sequence of PWLCM is used to generate a random sequence. Indices of the ordered generated sequence were used to embed the encrypted message in randomly selected audio samples. The encrypted data were embedded on the higher layers other than the LSB using efficient bits adjustment algorithm, in order to increase the robustness against noise addition or MPEG compression. The use of chaotic map for generating the keys gave their method more strenght.

In the same year 2017, Ahmed et al [17] suggested two approaches to speed up Serpent. The two approaches' main feature is to encrypt/decrypt 512 bit input with 256 bit S-Box and 256 bit round key with just more dividing and mixing functions. Their main problem is in the 256 bit S-box which needs more time and hence did not succeed in speed test.

In March 2018, Elkamchouchi et al [18] used chaotic mapping and cycling group instead of S-box to improve Serpent's speed. They reduced the number of rounds to 10, and hence making Serpent faster than normal and even AES. On the other hands, in security verifications, chaotic map adds more strength to the algorithm by adding additional complexity to the key.

2018, Shah et all [19] reduce the number of rounds to 22 and use $4 * 4$ S-box. Their improved Serpent achieves 31% faster than normal but pays the cost on security.

Depends on these proposed modifications parallel execution and hardware implementation were the solution for speeding up Serpent algorithm, but hardware implementation adds limitation to the Serpent because its need additional device, so parallel execution become better but it decrease the security of the algorithm, this problem can be resolved by block key generation that has many ways to generate key for every block, the best way was chaotic map that used in image encryption because its faster and secure, so the good solution for enhancing Serpent algorithm is running it in parallel mode and generate key for every block using chaotic map.

3. **Proposed Key Generation Methods.**

Hereby, the following sections explain two proposed key generation methods. Our proposed method was tested using the two proposals and a comparison of the time execution was performed.

Algorithm 1: Lorenz 96 Key Generator

```
Input: key: encryption key in array of 32 bytes
       blockno: Number of blocks > 1
Output: (byte)keylist: Convert Integer array to Byte array
1  $x \leftarrow (int)key$ /* Convert byte array to Integer array */
2  $keylist \leftarrow newarray[blockno]$
3  $N \leftarrow length(key)$
4  $keylist[0] \leftarrow (x[1] - x[N-2]) * x[N-1] - x[0] + 8$
5  $keylist[1] \leftarrow (x[2] - x[N-1]) * x[0] - x[1] + 8$
6  $keylist[N-1] \leftarrow (x[0] - x[N-3]) * x[N-2] - x[N-1] + 8$
7  for $i = 2$ to $N-1$ do
8      $keylist[i] \leftarrow (x[i+1] - x[i-2]) * x[i-1] - x[i] + 8$
9  $output \leftarrow (byte)keylist$ /* Convert Integer array to Byte array */
10 return $output$
```

3.1. **Generating Encryption Keys using Lorenz 96 Chaotic Map.**

The first algorithm of chaotic maps was proposed in 1989 by Robert and Matthews by investigating Logistic map based on one-dimensional non-linear iterative function. Since then, many chaotic encryption algorithms were proposed that can be classified into five types [20][21]:

1. Pure position scrambling.
2. Open loop based chaotic synchronization stream cipher.
3. Chaotic self-synchronization stream cipher whose ciphertext is fed back into encryption process.
4. Chaotic self-synchronization stream cipher whose ciphertext is fed back into chaotic system.

5. Position scrambling combined with chaotic synchronization stream cipher or chaotic self-synchronization stream cipher.

Because of speed and low memory, chaotic maps were used in various areas such as random key generation. Hence, Lorenz 96 chaotic map is used to generate the block key by using the previous block key (or the user key if it ran on the first block) as input and convert it to an array of 32 bytes to generate new array of 32 bytes which will be the key of the next block as in algorithm 1 [22][23][24].

It uses multiple pseudo numbers based on multiple numbers as input by utilizing the simple mathematical functions explained hereby [25]:

$$Result[i] = (input[i+1] - input[i-2]) * input[i-1] - input[i] + 8$$

Algorithm 2: Proposed Block Key Generation (BKG)

```
Input: key: Encryption key = 256 bits
       blockno: Number of blocks > 1
Output: keylist: an array of keys
1 Function BKG($key, blockno$):
2     $keylist \leftarrow newarray[blockno]$
3     $currkey \leftarrow key$
4     for $i$ = 0 to blockno do
5         $currkey \leftarrow permutation(currkey)$
6         $currkey \leftarrow currkey \oplus key$
7         $keylist[i] \leftarrow SHA256(currkey)$
8     return $keylist$
```

3.2. **Generating Encryption Keys using BKG.** Block key generation (BKG) algorithm is proposed by Singh et al [14] that generates key for each block in three steps as shown in figure 3 and algorithm 2. Its operation is elucidated hereby:

1. the previous block key (or the user key if it runs on the first block) will be run in a Permutation function,
2. the output of the Permutation function will be passed to an exclusive OR function with the user key.
3. the output of exclusive OR function will be processed on the hash function SHA-256 that returns a 256 bit key which will be used for one block.

This algorithm will be compared with Lorenz 96 BKG that is part of the proposed algorithm to evaluate the better performance.

4. **The Proposed Method.** The proposed method tries to enhance Serpent algorithm by taking the advantage of ECB mode where every block encryption/decryption runs independently, i.e. to take the advantage of parallel computing which is used by Pendli et al [1] to enhance the AES performance.

A mode of operation describes how to repeatedly apply a cipher's single-block encryption/decryption to securely encrypt/decrypt amounts of data larger than an input block size (e.g. 128 bit in Serpent). ECB was the first proposed mode where the encryption of each block is independent and the output of any block does not affect the output of the other blocks. This makes ECB a faster mode with the ability of encrypting/decrypting multiple blocks at the same time but it does not hide data patterns. Thus, it does not provide cryptographic diffusion [14][27].

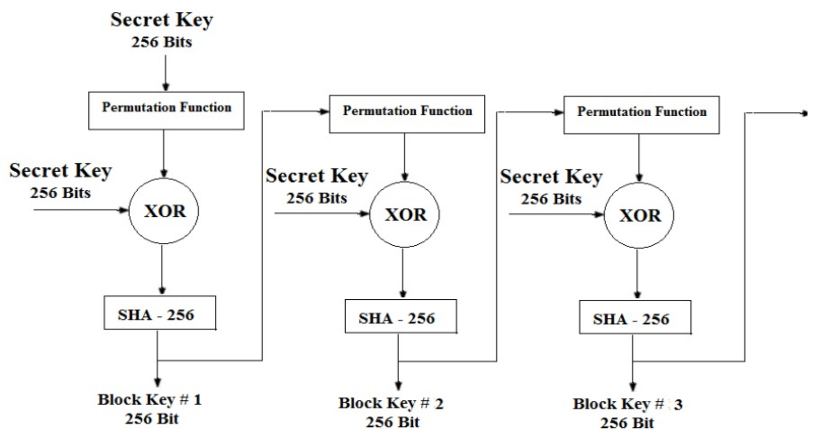


Figure 3. Remodeled BKG Algorithm from [14]

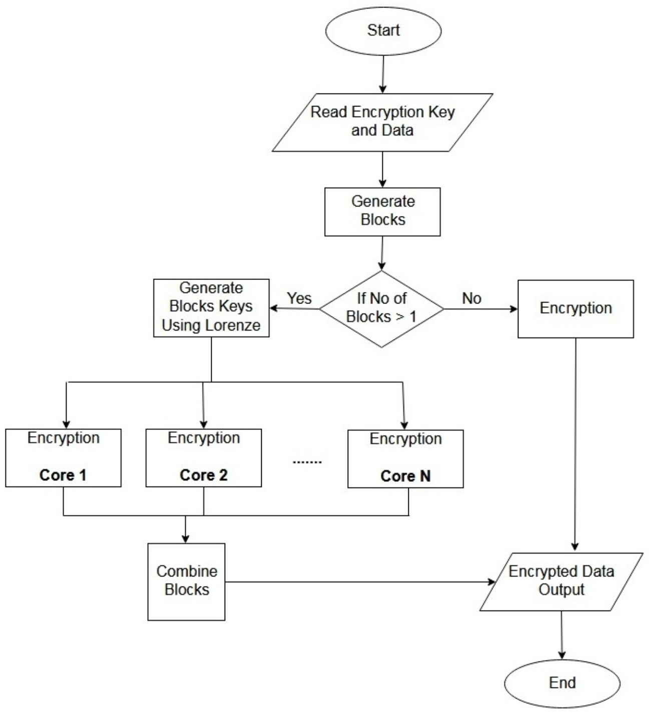


Figure 4. Flow chart for the proposed method

Furthermore, the proposed method uses Lorenz 96, which is one of chaotic algorithms, to generate 256 bit key for each block using single 256 bit user key as input to resolve the data pattern problem in ECB. The steps are depicted in figure 4 and algorithm 3.

The decryption is similar to the encryption process by calling the Serpent decryption function with the reverse order of the keys.

The proposed method has the following main steps:

4.1. **Reading input data.** In this step, the method starts with reading the user key and the input data that needs encryption or decryption from the user. The size of this key must be 256 bit (Serpent key size), if it is less than 256 bit then the method adds 0's to complete the length. Otherwise, if the length is more than 256 bit, the system uses the first 256 bit as key.

Algorithm 3: Proposed Method Algorithm

```
Input: key: Encryption key
       txt : Input data to be encrypted
       LorenzOrBKG: key generation is either through "Lorenz" or "BKG"
1  if $length(key) \neq 256$ then
2      $setsize(key, 256)$ /* Add 0's if less than 256 or get first 256 */
3  $blocks \leftarrow split(txt, 128)$ /* split input data into array of 128 bit block */
4  if $length(blocks) > 1$ then
5      if Lorenz then
6          $keylist \leftarrow Lorenz(key, length(blocks))$ /* call Lorenz algorithm */
7      else
8          $BKG(key, blockno)$ /* call BKG algorithm */
9      for $i = 0$ to $length(blocks)$ do
10         $encblock[i] \leftarrow SerpentEnc(blocks[i], keylist[i])$
11     encdata ← combine(encblock)
12 else
13     $encdata \leftarrow SerpentEnc(blocks[0], key)$
```

4.2. **Splitting the data.** Then the proposed method split the input data into fixed-length blocks depending on the encryption/decryption function block size. Here, the length of every block is 128 bit (Serpent block size). If the size of the last block is less than 128 bit, the method adds 0's to complete the length.

4.3. **Lorenz 96 key Generation.** In this step, the method checks if the number of blocks that is generated from the input data is one block, then there is no need for key generation algorithm to generate more keys and the method skips this step. Otherwise, the method uses Lorenz 96 chaotic map to generate encryption keys.

4.4. **Parallel Serpent.** Next, Serpent algorithm is run to encrypt/decrypt each block independently with its own key. This is done by distributing the blocks of encryption/decryption process in the available processors to get advantage of parallel computing.

4.5. **Uniting the blocks.** In the last step of the proposed method, after all encryption/decryption processes are complete, the method combines the encrypted/decrypted blocks into a single output data.

5. Experimental Results and Analysis.

The implementation results are reported in this section. It contains comparisons between the execution time of

- sequential and parallel implementation for Serpent block cipher,
- Serpent using BKG and the proposed method Serpent using Lorenz 96 BKG in sequential and parallel implementations.

Serpent, Serpent using BKG, and Serpent using Lorenz 96 BKG were implemented in Windows 7 Ultimate Service Pack 1 OS with 4.00 GB RAM using JAVA programming language with JDK version 1.8.0 and JAVA Heap 1 GB. For parallelization, ForkjoinPool framework was used.

5.1. **Sequential Execution Results.** Table 1 and figure 5 show the sequential execution results. There is no significant execution time difference between Serpent using Lorenz 96 BKG and Serpent. Actually, the execution of Serpent using BKG needs more time than both Serpent with Lorenz 96 BKG or traditional Serpent, which indicates that the proposed method, Serpent with Lorenz 96 BKG, has achieved less execution time than Serpent using BKG.

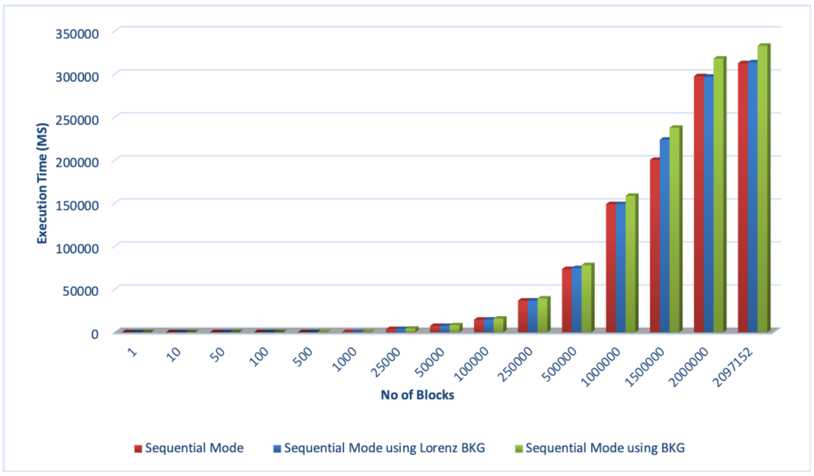


FIGURE 5. Execution Time for Serpent, Serpent using BKG and Serpent using Lorenz 96 BKG in Sequential Mode

TABLE 1 Execution Time (in msec) of Serpent, Serpent using BKG and Proposed Method,

# Blocks	Sequential Implementation			Parallel Implementation		
	Classical Serpent	Serpent using BKG	Serpent using Lorenz 96	Classical Serpent	Serpent using BKG	Serpent using Lorenz 96
1	0.7	0.7	0.7	0.7	0.7	0.7
10	15	31	16	15	47	31
50	46	78	46	47	78	46
100	62	94	62	62	94	62
500	140	234	156	140	219	125
1000	218	327	234	203	297	187
25000	3728	4197	3728	1919	2418	1934
50000	7394	8099	7332	3479	4326	3546
100000	14612	15827	14690	6911	8256	6936
250000	36870	39302	36948	16950	19793	18159
500000	73565	78013	74564	34023	36125	34414
1000000	149007	158610	148949	67448	80215	69890
1500000	200391	237823	223996	102556	120050	102499
2000000	297730	318203	297207	133559	159743	139229
2097152	312787	333166	313888	147325	153504	148565

5.2. **Parallel Execution Results.** Parallel execution results are shown in figure 6 and the last three columns of table 1. The Serpent using Lorenz 96 necessitated more time than classical Serpent because of the block key generation using Lorenz 96 chaotic map. But Serpent using BKG required more time than both traditional Serpent and the proposed method. This shows generating the keys using Lorenz 96 chaotic map had less execution time than BKG for the Serpent algorithm in addition to the more added security.

According to the above results, parallel execution is better than sequential execution, specifically if the number of blocks is greater than 1000 blocks, otherwise the difference becomes in the range of 10 msec which does not have a big effect. Specifically, the reduction rate for bigger number of blocks is in the range of 48.1 - 53.2%.

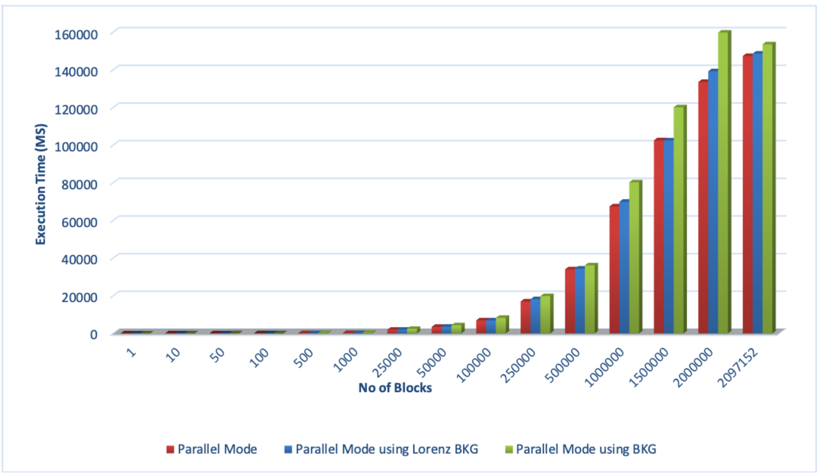


FIGURE 6. Execution Time for Serpent, Serpent using BKG and Serpent using Lorenz 96 BKG in Parallel Mode

5.3. **Sequential versus Parallel Execution Results.** Comparisons between sequential and parallel execution are presented in figures 7, 8 and 9. Figure 7 shows the execution of Serpent in sequential and parallel modes. Whilst 8 presents the execution of Serpent using BKG in sequential and parallel modes. Finally, the execution of Serpent using Lorenz 96 BKG in sequential and parallel modes is illustrated in figure 9.

All three graphs show that the parallel execution time is less than the sequential when the number of blocks is large. On the other hand, if the number of blocks is less than 1000 blocks, the difference in execution time is insignificant.

5.4. **Serpent versus Serpent using Lorenz 96 BKG Execution.** The comparison between Serpent in sequential and parallel modes and Serpent using Lorenz 96 chaotic-based BKG in parallel mode is illustrated in figure 10.

The execution time of sequential mode of Serpent using secure operation mode such as CBC and others is very high compared to Serpent using Lorenz 96 BKG or Serpent in parallel mode. Although the difference in execution time between Serpent in parallel mode and Serpent with Lorenz 96 chaotic-based BKG is not significant, the generation of the block keys using Lorenz 96 chaotic map provides the needed security in ECB mode.

5.5. **Brute Force Attack.** The results of the brute force attack is illustrated in table 2, which is expressed by the number of probability of breaking the key. It is clear that the number of possibilities for breaking the key is large for the novel enhanced Serpent using Lorenz 96 BKG algorithm compared to the traditional Serpent. Actually, it could be calculated as follows:

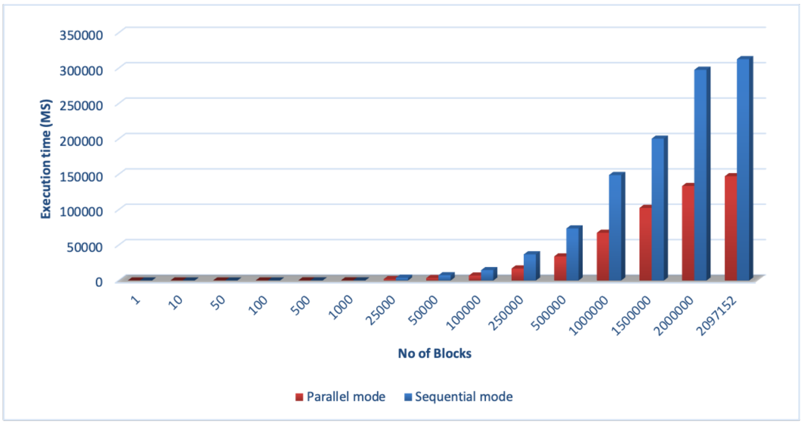


FIGURE 7. Execution Time for Serpent Algorithm in Sequential and Parallel Modes

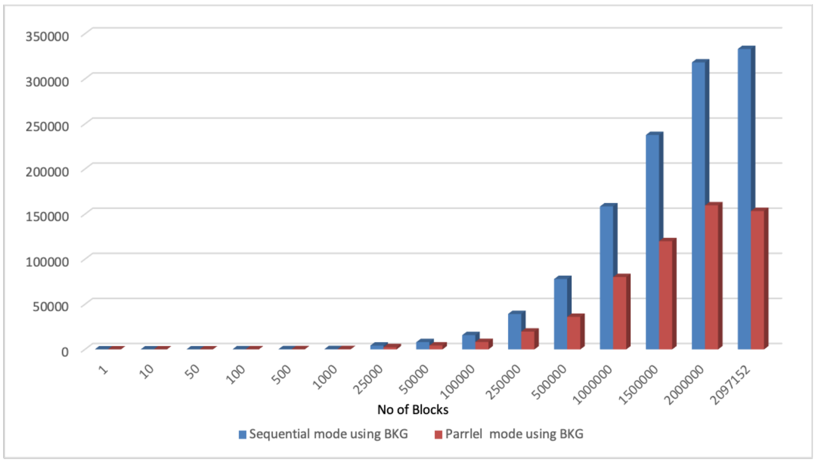


FIGURE 8. Execution Time for Serpent using BKG in Sequential and Parallel Modes

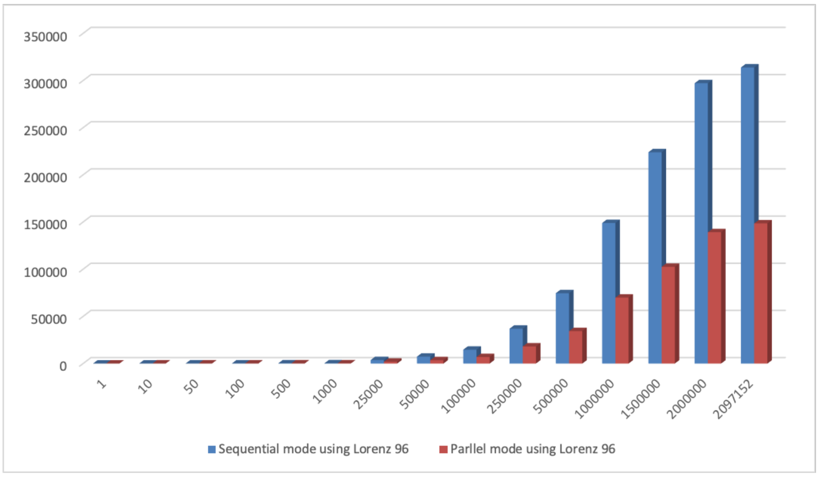


FIGURE 9. Execution Time for Serpent using Lorenze BKG in Sequential and Parallel Modes

$$n \times 2^{256} \qquad \text{where } n \text{ is the block number}$$

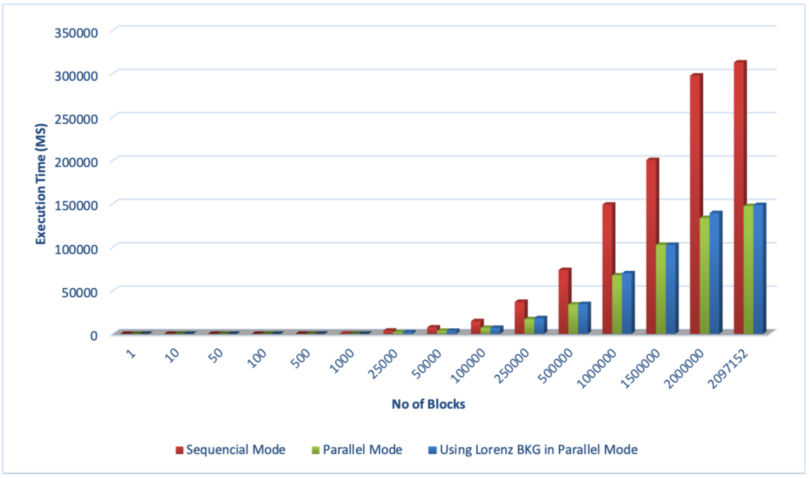


FIGURE 10. Execution Time for Parallel Serpent, Sequential Serpent and Serpent using Lorenz 96 BKG in Parallel Mode

TABLE 2 Comparison of Key Space Between Serpent and Serpent using BKG and Serpent using Lorenz 96 BKG based on Brute Force Attack

Number of blocks	**Serpent**	**Serpent using BKG**	**Serpent using Lorenz 96 BKG**
1	2^{256}	1×2^{256}	1×2^{256}
10	2^{256}	10×2^{256}	10×2^{256}
50	2^{256}	50×2^{256}	50×2^{256}
100	2^{256}	100×2^{256}	100×2^{256}
500	2^{256}	500×2^{256}	500×2^{256}
1000	2^{256}	1000×2^{256}	1000×2^{256}
25 000	2^{256}	$25\ 000\times2^{256}$	$25\ 000\times2^{256}$
50 000	2^{256}	$50\ 000\times2^{256}$	$50\ 000\times2^{256}$
100 000	2^{256}	$100\ 000\times2^{256}$	$100\ 000\times2^{256}$
250 000	2^{256}	$250\ 000\times2^{256}$	$250\ 000\times2^{256}$
500 000	2^{256}	$500\ 000\times2^{256}$	$500\ 000\times2^{256}$
1000 000	2^{256}	$1000\ 000\times2^{256}$	$1000\ 000\times2^{256}$
1 500 000	2^{256}	$1\ 500\ 000\times2^{256}$	$1\ 500\ 000\times2^{256}$
2 000 000	2^{256}	$2\ 000\ 000\times2^{256}$	$2\ 000\ 000\times2^{256}$
2 097 152	2^{256}	$2\ 097\ 152\times2^{256}$	$2\ 097\ 152\times2^{256}$

Table 2 shows that the key space of Serpent using BKG and Serpent using Lorenz 96 BKG is the same which is very large compared to that key space of the traditional Serpent algorithm. This means a stronger key and hence higher security which proves the efficiency of the proposed algorithm, as its strength depends on the large key space, thus making the brute force attack more difficult.

6. **Comparison with State-of-the-Art.** The proposed Serpent with Lorenz 96 key generation is further compared with reference to the reduction in execution time with the works of Pendli et al [1] and Rahmah et al [28]. Blatantly, as demonstrated in table 3, it is clear that the proposed Serpent with 96 key generation prevails existing enhanced Serpent

schemes by achieving a reduction rate of 48.1 - 53.2% whilst [1] attained a reduction of 40 - 45% and [28] a 20.6 - 23.8% reduction in execution time.

Table 3 Comparison of the Time Execution Reduction Rate in % for different Serpent Enhanced Schemes

Reference	**Time Execution Reduction rate in %**
Proposed Method	48.1 - 53.2%
Pendli et al [1] 2016	40 - 45%
Rahmah et al [28] 2020	20.6 - 23.8%

7. **Conclusion.** A novel approach enhancing the performance and security of the Serpent algorithm is proposed. The Serpent is run in parallel mode and a key is generated for every block using Lorenz 96 chaotic map. Based on the experimental results, it is concluded that the parallel implementation of Serpent is an appropriate method when the performance is the main concern. Moreover, adding Lorenz 96 BKG to Serpent algorithm with EBC mode is an excellent option when high security and performance is needed.

The experimental results got the maximum execution time for the Serpent with BKG and Serpent with Lorenz 96 BKG because of the need for generating block keys. This can be solved by generating all keys before encryption. To reduce this time, it becomes a good idea to start parallel encryption after generating the first key. Furthermore, it is better to use sequential mode like CBC mode for a few number of blocks. Moreover, the proposed method was compared with existing enhanced Serpent schemes and proved its efficacy in reducing the execution time by 53.2%. Thus, the results proved the effectiveness of the proposed novel approach and the Serpent's performance and security are significantly improved.

As a future work, we recommend to test the proposed method using CTR instead of EBC cipher mode.

REFERENCES.

[1] Pendli, Vandan and Pathuri, Mokshitha and Yandrathi, Subhakar and Razaque, Abdul, *Improving performance of Advanced Encryption Standard algorithm*, Mobile and Secure Services (MobiSecServ), 2016 Second International Conference on, 1–5, IEEE, 2016.

[2] Rijmen, Vincent and Daemen, Joan, *Advanced encryption standard*, Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology, 19–22, 2001.

[3] Izevbizua, Peter OdionData Security in the Cloud , *using Serpent Encryption and Distributed Steganography*, European Scientific Journal, ESJ, 11(18), 2015.

[4] Anderson, Ross and Biham, Eli and Knudsen, Lars, *Serpent: A candidate block cipher for the Advanced Encryption Standard*, Página oficial do SERPENT, disponível em http://www. cl. cam. ac. uk/˜ rja14/serpent. html, 2005.

[5] Tayel, Mazhar and Dawood, George and Shawky, Hamed. A Proposed Serpent-Elliptic Hybrid Cryptosystem For Multimedia Protection. *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 387–391,IEEE (2018).

[6] Simha S., Prathibha and Prof. Priya, Hari, *Enhancing Cloud Security with the Implementation of Serpent Encryption Algorithm*, Imperial Journal of Interdisciplinary Research, 3(5), 2017.

[7] Mohammadreza Naeemabadi, Behnam Sadeghi Ordoubadi, Alireza Mehri Dehnavi and Kambiz Bahaadinbeigy. Comparison of Serpent, Twofish and Rijndael encryption algorithms in teleophthalmology system. *Advances in Natural and Applied Sciences, 9(4) April 2015, Pages: 137-149*. 2015.

[8] Nagendra, M and Sekhar, M Chandra, *Performance improvement of Advanced Encryption Algorithm using parallel computation*, International Journal of Software Engineering and Its Applications, 8(2), 287–296, 2014.

[9] Osvik, Dag Arne, *Speeding up Serpent*, AES Candidate Conference, 317–329, Citeseer, 2000.

[10] Najafi, B and Sadeghian, B and Zamani, M Saheb and Valizadeh, A, *High speed implementation of Serpent algorithm*, Microelectronics, 2004. ICM 2004 Proceedings. The 16th International Conference on, 718–721, IEEE, 2004.

[11] Fouda, JS Armand Eyebe and Effa, J Yves and Sabat, Samrat L and Ali, Maaruf, *A Fast Chaotic Block Cipher for Image Encryption*, Communications in Nonlinear Science and Numerical Simulation, 19(3), 578–588, Elsevier, 2014.

[12] Taher, Mai Hossam and El_Deen, Ali E Taki and Abo-Elsoud, Mohy E, *Hardware Implementation of The Serpent Block Cipher Using FPGA Technology*, Journal Impact Factor, 5(10), 34–44, 2014.

[13] Ali, Yossra Hussain and Ressan, Haider Aabdali, *Image Encryption Using Block Cipher Based Serpent Algorithm*, Engineering and Technology Journal, 34 (2 Part (B) Scientific), 278–286, University of Technology, 2016.

[14] Singh, Harpreet and Singh, Paramvir, *Enhancing AES using Novel Block Key Generation Algorithm and Key Dependent S-boxes*, Cyber-Security and Digital Forensics, 30, 2016.

[15] Altigani, Abdelrahman and Abdelmagid, Muawia and Barry, Bazara, *Analyzing the Performance of the Advanced Encryption Standard Block Cipher Modes of Operation: Highlighting the National Institute of Standards and Technology Recommendations*, Indian Journal of Science and Technology, 9(28), 2016.

[16] Alwahbani, Samah MH and Elshoush, Huwaida TI, *Chaos-based Audio Steganography and Cryptography Using LSB Method and One-Time Pad*, Proceedings of SAI Intelligent Systems Conference, 755–768,Springer, 2016.

[17] Prof. Kadhim, Alaa F. and Hassin, Semaa and Ali, Gada and Ahmed, Israa, *New Approach for Serpent Block Cipher Algorithm Based on Multi Techniques*, Iraqi Journal of Information Technology, 7(3),1–13, Iraqi Association of Information, 2017.

[18] Elkamchouchi, Hassan M and Takieldeen, Ali E and Shawky, Mahmoud A, *A Modified Serpent Based Algorithm for Image Encryption*, National Radio Science Conference (NRSC), 2018 35th, 239–248, IEEE, 2018.

[19] Shah, Tariq and ul Haq, Tanveer and Farooq, Ghazanfar, *Serpent Algorithm: An Improvement by* 4×4 *S-Box from Finite Chain Ring*, 2018 International Conference on Applied and Engineering Mathematics (ICAEM), 1–6, IEEE, 2018.

[20] Matthews, Robert, *On the Derivation of a "Chaotic" Encryption Algorithm*, Cryptologia, 13(1),29–42,Taylor & Francis, 1989.

[21] Lin, Zhuosheng and Wang, Guangyi and Wang, Xiaoyuan and Yu, Simin and Lü, Jinhu, *Security Performance Analysis of a Chaotic Stream Cipher*, Nonlinear Dynamics, 1–15, Springer, 2018.

[22] Audhkhasi, Kartik, *Chaos Based Cryptography*, CiteSeerX Scientific Literature Digital Library and Search Engine, 2009.

[23] Kocarev, Ljupco, *Chaos-based Cryptography: A Brief Overview*, IEEE Circuits and Systems Magazine, 1(3),6–21, IEEE, 2001.

[24] Xiao, Di and Liao, Xiaofeng and Deng, Shaojiang, *One-way Hash Function Construction Based on the Chaotic Map with Changeable-parameter*, Chaos, Solitons & Fractals, 24(1), 65–71, Elsevier, 2005.

[25] Lorenz, Edward N, *Predictability: A problem partly solved*, Proc. Seminar on predictability, 1(1), 1996.

[26] Elshoush HT, Al-Tayeb BM, Obeid KT. 2021. Enhanced Serpent algorithm using Lorenz 96 Chaos-based block key generation and parallel computing for RGB image encryption. *PeerJ Comput. Sci.* 7:e812 http://doi.org/10.7717/peerj-cs.812

[27] Diedon Bujari and Erke Aribas. Comparative Analysis of Block Cipher Modes of Operation. *International Advanced Researches & Engineering Congress*, 2017 http://iarec.osmaniye.edu.tr/. Osmaniye/TURKEY 16-18 November 2017.

[28] Hassan Rahmah Zagi , Abeer Tariq Maolood, *A Novel Serpent Algorithm Improvement by the Key Schedule Increase Security*, Tikrit Journal of Pure Science.