International Journal of Electrical and Computer Engineering (IJECE)
Vol. 14, No. 6, December 2024, pp. 6753~6772
ISSN: 2088-8708, DOI: 10.11591/ijece.v14i6.pp6753-6772

Text encryption using secure and expeditious multiprocessing $Serpent_{CTR}$ using logistic map

Huwaida T. Elshoush[1], Duaa M. Ahmed[1], Abdalmajid A. Ishag[1], Muawia A. Elsadig[2], Abdelrahman Altigani[3]
[1]Computer Science Department, Faculty of Mathematical Sciences and Informatics, University of Khartoum, Khartoum, Sudan
[2]College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, Dammam, Saudi Arabia
[3]Computer Information Science, Higher Colleges of Technology, Al Ain, United Arab Emirates

Article Info

Article history:

Received Dec 10, 2023
Revised Jul 14, 2024
Accepted Aug 6, 2024

Keywords:

Cryptography
CTR mode
Logistic map
Multiprocessing
Parallel computing
Serpent
Text encryption

ABSTRACT

Unarguably performance is a critical factor to the success of any cipher. Al-Beit Serpent is more secure than advanced encryption standard (AES), it faces limitations such as speed and memory requirement. Hence, this paper proffers a text encryption method $Serpent_{CTR-LogisticMap}$ that ameliorates the performance by running Serpent in parallel using the counter (CTR) encryption mode and further enhances the security by generating sub-keys for each block using logistic map. The intricate logistic map generated keys adds robustness to the proposed algorithm. Comprehensive experiments using Python 3.9 on commonly used metrics verify the efficacy of the proposed method in terms of execution time, central processing unit (CPU) usage, security analysis including key space, strict avalanche effect and its randomness. The encryption/decryption reduction rate reached up to 80.81%. It is worthy of note that it is effectually resistant to brute force attacks having a large key space in addition to its dependency on the number of blocks besides the randomly generated keys. The enhanced Serpent was examined using the statistical test suite (STS) recommended by the National Institute of Standards and Technology (NIST) and verified its randomness by passing all tests. Furthermore, it efficaciously resisted statistical analysis, particularly histogram and correlation coefficient analysis. Moreover, it prevails over current methods when juxtaposed with them in terms of performance, key space, key sensitivity, avalanche effect, histogram analysis and correlation coefficient, ergo affirming its efficiency.

Corresponding Author:

Huwaida T. Elshoush
Computer Science Department, Faculty of Mathematical Sciences and Informatics, University of Khartoum
Khartoum, Sudan
Email: htelshoush@hotmail.com

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]. In 1997, the US National Institute of Standards and Technology [2] issued that they need to choose an alternative to data encryption standard (DES); hence appeared the chosen alternative which is known as advance encryption standard (AES). Many algorithms were proposed as candidates, and Serpent, which is also a symmetric block, was one of the AES competition finalists. Serpent and Rijndael-AES winner- have many similarities; the main difference is that Rijndael is faster but Serpent is more secure [2]–[5]. Serpent runs on four 32-bit words (128-bits) data block and three different key sizes, namely 128, 192, or 256 bits. It has 32 rounds, where each round utilizes one of eight 4×4 S-boxes. Its functions were developed to be implemented in parallel, using 32 bit slices, hence boosting parallelism [6]–[16]. The nonlinear layer in AES uses an 8×8 S-box whereas Serpent uses eight different 4×4 S-boxes. The 32 rounds means that Serpent has a higher security margin than Rijndael; however, Rijndael with 10 rounds is faster and easier to implement for small blocks, as the 32 rounds of Serpent make it a bit slower and complex to implement on small blocks. Thus, Serpent and Rijndael are somewhat similar; the main difference is that Rijndael is faster (having fewer rounds) yet Serpent is more secure. Compared to 3DES, Serpent is easy to implement, fast and more secure [14]–[17]. Serpent uses 256 bit key to encrypt 128 bit of plaintext in three main functions, namely initial per- mutation (IP), round function (R) and final permutation (FP) [18]. It is secure, and specifically has suitable functioning for protecting against power and timing attacks [19]. Yet, it faces limitations such as memory requirement and execution time. The number of rounds in Serpent, which is 32, affect the performance directly. The proposed approach enhances the execution time of Serpent by using parallel computing to speed-up encryption and decryption process. Furthermore, the proposed approach enhances the security by generating sub keys for each block using Logistic Map block key generation algorithm [20]–[29]. Moreover, the algorithm is run in counter (CTR) mode [30]–[34].

This work is an extension of research [18]. The inauguration of this research is entirely to improve the performance as well as the security of Serpent algorithm. Hence, the proposed method enhanced the execution performance time by running Serpent in CTR using multi-processes. Furthermore, logistic map is utilized to generate block keys randomly and hence boosting the security of the proposed method. Additionally, it is juxtaposed with prevailing methods and traditional Serpent and proved its efficiency. The contribution of this research can be summarized hereafter: i) The Serpent is expedited by splitting the plaintext input into blocks, and moreover generating sub-keys using logistic map, and finally running the proposed algorithm in CTR mode; ii) Using logistic map boost the proposed algorithm from the security facet as a consequence of the sub-key complexity and randomness. Moreover, different sub-keys for each block disguise the plaintext patterns; iii) The performance of the proposed method was tested and evince its efficient speed and security; and iv) Furthermore, when juxtaposed with traditional Serpent and current schemes, the proposed method surpassed them and gave favorable and supreme results.

The remainder of the paper is structured as follows: The recent schemes in improving the performance and security of the Serpent are explored in section 2. Section 3 elucidates the proposed method, together with its detailed algorithms. The results and discussion are presented in section 4. Finally, section 5 concludes the paper as well as recommending some future work.

2. RELATED WORK

Hereafter, a review of some research attempts to improve the speed of Serpent. Some researchers such as [35]–[38] improved Serpent by changing the functionality of the algorithm. For instance, researchers [35] modified the original S-box to consume less time. In a similar fashion, researchers [36] also modified Serpent by modifying S-box. They use 4×4 S-box consisting of bytes instead of nibbles and achieved less speed by 16.54% than the tradition Serpent. On the other hand, researchers [37], [38] utilize chaotic map to enhance Serpent. In particular, Elkamchouchi *et al.* [37] replaced the S-box with chaotic mapping and cycling group and reduced the number of rounds to 10. While Yousif [38] exchanged the static permutation and substitution with dynamical properties using logistic chaos map, hence yielding great randomness when juxtaposed with traditional Serpent. The use of chaotic map enhanced the security and Serpent became more robust. Likewise, studies [39] and [40] considered the use of chaotic map in text encryption. Particularly, Ekhlas *et al.* [39] utilized chaotic to encrypt text files by diffusing the positions of the plaintext ASCII values. Their method has a large key space, a uniform histogram and is sensitive to any bit altering in text or key. Whereas, the work of Charalampidis *et al.* [40] presented a novel 1-D chaotic map that displays zones of constant chaos, and high values of Lyapunov exponent to generate a pseudorandom bit generator. Zagi and Maolood [41] suggested a new design to the key generation as the security of any encryption algorithm relies solely on the security of the generated keys. Singh and Singh [42] introduced the idea of running each block in parallel and generating different keys for every block.

In a recent research, Elshoush *et al.* [43] proffered running the Serpent in parallel and further generating block keys using Lorenz 96 chaotic map. Their method attained a reduction of 53.2% compared to classical Serpent, whereas preliminary version of the proposed method in [18] achieved up to 91% when running in five processes as they used CTR mode. Hussain *et al.* [44] modified Serpent by using power associative loop and group of permutations. Their technique's speed is comparable to 3-DES, and has higher security, sensitivity and is resistant to crypto analytic attacks.

3. RESEARCH METHOD
3.1. Key generation

The proposed $Serpent_{CTR-LogisticMap}$ method has two options for generating the key: automated key generation using the user key as input or utilizing logistic map. Logistic map is a one-dimensional discrete- time map that exhibits unpredicted degree of complexity [18]. The automatic generation is more secure and thus preferred as the key is generated randomly using Algorithm 1. In the first instance, two initial values x and r are randomly generated between [0,1] and [0,4] respectively to 3 decimal places. The value r then remains constant. Next, the key is generated after $KeySize + 1$ iterations to give a decimal number which after converting it to binary yields a 256-bit key.

Algorithm 1. Key generation using logistic map
```
Input: KeySize
Output: HexaKey   // Hexadecimal 256 Key
1. Function Logist icMap (KeySize):
2.   Xo ← randomnumber (0, 1), 3;
       // generating an initial value X0 between 0 and 1 (to 3 decimal places)
3.   r ← randommunber(0,4),3;4
       //generating a positive constant r between 0 and 4 (to 3 decimal places)
4.   HexaKey ← empty
5.   for i=0 to KeySize do
6.      X(i+1) ← r * Xi(1 − Xi)     // Logistic map equation
7.      HexaKey ← HexaKey.append(ConvertDecToHexa(DecValueOf (X*10^16)/256));
       //Each time get one Hexadecimal Value
8.   End
9.   Return: HexaKey
```

3.2. Encryption/decryption using the proposed $Serpent_{CTR-LogisticMap}$

The proposed $Serpent_{CTR-LogisticMap}$ encryption/decryption Algorithm 2 runs the Serpent using CTR mode, and allows it to encrypt/decrypt multiple blocks in parallel using multiple CPUs, and that is done by dividing the input plaintext into data chunks. Each chunk will be handled by one CPU and the data chunk size is based on the number of CPUs. Hence, multiple CPUs will be running with part of an initial input, and handles the input the same way that the normal Serpent works. Algorithm 2 takes as input the $InputText$ to be encrypted/decrypted, a Key generated as explained previously in Algorithm 1, and initial vector IV and $ConcProcessed$ which is the number of concurrent processes. First, it calls Algorithm 3 $SplitBlocks$ $(BlockSize, InputText)$ with parameters $blockSize$ and $InputText$ to split the text into blocks to be processed expeditiously using multiple CPUs and returns SplitBlocks, which is a list of split blocks. In the first instance, Algorithm 3 $SplitBlocks(BlockSize, TextToSplit)$ takes as input parameters $BlockSize$, and the $TextToSplit$. First, it checks if the text to be split is divisible by the block size, and finally returns a list of split up blocks $SplitBlocks$, to be passed back to algorithm 2. This is depicted in the flowchart shown Figure 1. Next, Algorithm 2 continues to prepare the list of blocks to be scheduled for each CPU. It then calls Algorithm 4 $GenerateNonce$ with parameters IV and $TaskDataIndex$, which is the index of the list of blocks. Finally, Algorithm 4 returns a Nonce for each CPU to be used as an input to Algorithm 5 $BulkEncDec$ with the other two parameters $TaskBlocks$ and Key. Subsequently, Algorithm 5 encrypt/decrypt each block using the proposed Serpent using logistic map and CTR mode. Thereafter, the encrypted/decrypted blocks are joined together to form the final cipher-text/plaintext to be returned as $OutputText$ by the main Algorithm 2. It is worthy of mention that when decrypting, cipher text data will be split and at the end, the produced blocks are amalgamated together to construct the final plaintext.

Algorithm 2. Proposed $Serpent_{CTR-LogisticMap}$ encryption/decryption process
```
Input: InputText, Key, IV, ConcProcesses
// InputText = plaintext/ciphortext; Key=256-bits user key; IV=64-bits initial-vector;
ConcProcesses no. of concurrent processes
Output: OutputTert,
 1.  Function EncDeCTR (InputText, Key, IV,ConcProcesses) :
 2.    BlockSize ← 128
 3.    SplitBlocks ← call SplitBlocks (BlockSize, InputText)  // Calling method SplitBlocks
      with parameters Blocksize and InputText to be encrypted/decrypted & output SpluBlocks
 4.    if there is an error in SplitBlocks method then
 5.       go to step 30
 6.    end
 7.       DataSize ← length (SplitBlocks)
 8.       ChunkSizelnt ← DataSize/ConcProcesses
 9.    if ChunkSizelnt is 0 then
 10.      ChunkSizeInt ← 1
 11.   end
```

```
       //If the number of processes is more than number of blocks, then each process will
       handle one block
 12.   Task Datalndex ← 1
 13.   RunningProcessesHolder ← empty
 14.   while TaskDatalndex < DataSize do
 15.   TakeBlocks ← Split Blocks.take Blocks(from TaskDatalndex,..., to (TaskDatalndex +
     ChunkSizeInt))
 16.   Task Nonce ← call GenerateNonce(IV.Task Datalndex)    // Calling method GenenuteNoec
     with parameters TaskDataIndex & output variable Nonce
 17.   StartNewProcess ← call Bulk EncDec(Task Blocks, TaskNonce. Key)   // Calling mathod
     BulkEnDec with parameter TaskBlocks, TaskNonce and key & outputEncrDecTaskBlocks
 18.   RunningProcessesHolder.add (StartNewProcess)
 19.   Task DataIndex ← Task DataIndex - ChunkSizeInt   // update chunk index
 20.   end
 21.   Wait until all processes are done
 22.   RunningP rocessIndex ← 1
 23.   OutputText ← empty
 24.   NumberOfTasks ← length(RunningP rocessesHolder)
 25.   while RunningProcessIndex < NumberOfTasks do
 26.         OutputText.join(RunningProcessesHolder[RunningProcessIndex].result)
 27.         RunningProcessIndex ← RunningProcessIndex + 1   // next process index
 28.   end
 29.   Return: OutputText
 30. End Function
```

Algorithm 3. Splitting the blocks algorithm

```
Input: BlockSize, T extT oSplit;    // Expected block size is 128
Output: SplitBlocks;    // A list of split blocks
 1.  Function SplitBlocks(BlockSize, TextToSplit):
 2.  TextT oSplitBits ← length(TextT oSplit)
 3.  if TextToSplitBits mod BlockSize ≠ 0 then
 4.  raise error and go to step 14
 5.  end
     ;     // should be divisible by block size
 6.  SplitBlocks ← empty
 7.  BlockIndex ← 1
 8.  while BlockIndex < TextToSplitBits do
 9.  TakeBlockF romT ext ← TextToSplit.split(from BlockIndex to (BlockIndex + BlockSize))
 10. SplitBlocks.add(TakeBlockF romText)
 11. BlockIndex ← BlockIndex + BlockSize ;    // update next block start point
 12. end
 13. Return: SplitBlocks;    // List of split blocks
 14. End Function
```

Algorithm 4. Get Nonce by IV-base and ID algorithm

```
Input: IV base, ID;    // IV-base is 64 bits; ID is an integer
Output: Nonce
 1. Function GenerateNonce(IV base, ID):
 2.       NoncePostf ix ← ConvertDecTo64Bits(ID)
 3.       Nonce ← IV base.append(NonceP ostf ix)
 4.       Return: Nonce
 5. End Function
```

Algorithm 5. Bulk encrypt/decrypt algorithm

```
Input: TaskBlocks, TaskNonce, Key
Output: EncDecTaskBlocks
 1.  Function BulkEncDec(TaskBlocks, TaskNonce, Key):
 2.       TaskBlocksSize ← Length (TaskBlocks)
 3.       EncDecTaskBlocks ← empty
 4.       BlockIndex ← 1
 5.       while BlockIndex < TaskBlocksSize do
 6.          EncDecBlcok ←
               BinaryXOR(TaskBlocks.getBlockAt[BlockIndex], SerpentEncDec(TaskNonce, Key))
 7.          EncDecTaskBlocks.add(EncDecBlcok)
 8.          TaskNonce ← IncrementNonce(TaskNonce) ;    // Add 1 to the current nonce
 9.          BlockIndex ← BlockIndex + 1;    // Go to the next block
 10.        end
 11.      Return: EncrDecTaskBlocks
 12. End Function
```

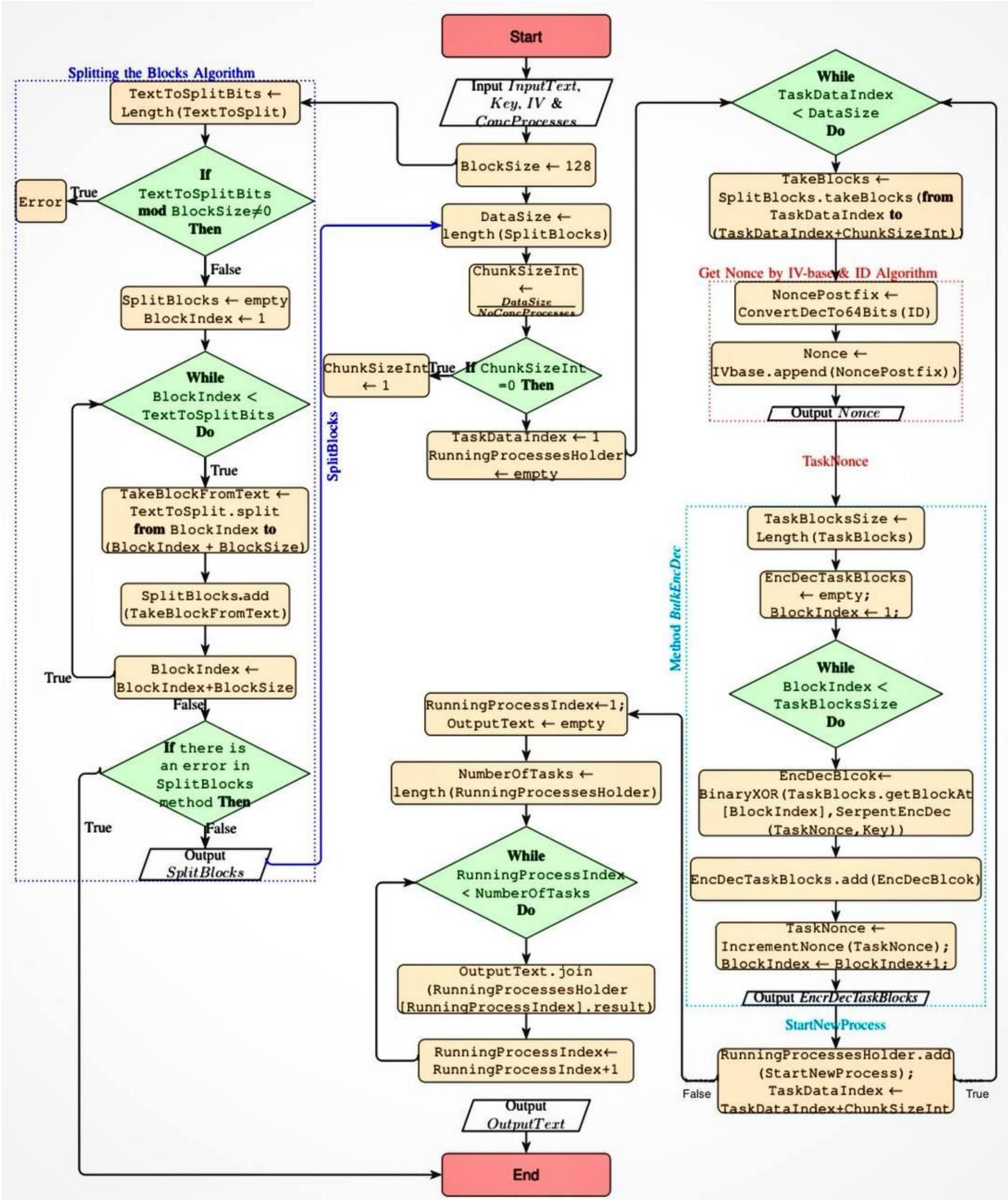


Figure 1. The proposed encryption/decryption $Serpent_{CTR-LogisticMap}$ algorithm flowchart

3. RESULTS AND DISCUSSION

This section presents series of experiments to evaluate the performance and demonstrating the efficiency of the proposed expeditiousness $\boldsymbol{Serpent}_{CTR-LogisticMap}$ in relation to the execution time, and comparing its performance with the traditional Serpent. From the security aspect, the key space of the proposed method has been investigated. Moreover, comparisons to related schemes are also conducted.

4.1. Preliminaries

The proposed $\boldsymbol{Serpent}_{CTR-LogisticMap}$ and traditional Serpent were implemented using Python 3.9. All the experimental results were tested on a laptop with Windows 64 bit OS. The laptop specifications were 4 GB RAM and Core i5 processor with 2.20 GHz speed.

4.2. Performance testing

The execution time is one of the most significant metrics that reflects the performance of any encryption algorithm. Hence, the performance of proposed method $Serpent_{CTR-LogisticMap}$ was tested using 10 different block sizes and different number of CPUs and the execution times were meticulously recorded in seconds. Each block size is tested twice for both encryption and decryption, hence a total of 80 experiments were performed using different processors yielding a grand total of 160 experiments. Furthermore, the performance of the proposed method is scrutinized and juxtaposed against the traditional Serpent as depicted in Table 1 and Table 2. These tables were split into two groups of small and large block sizes. The last three columns in each table expound the encryption/decryption reduction rates when compared with the traditional Serpent.

Table 1. Encryption execution time and reduction rate (in %) of proposed $\mathbf{S}erpent_{CTR-LogisticMap}$ compared with traditional Serpent for different block sizes

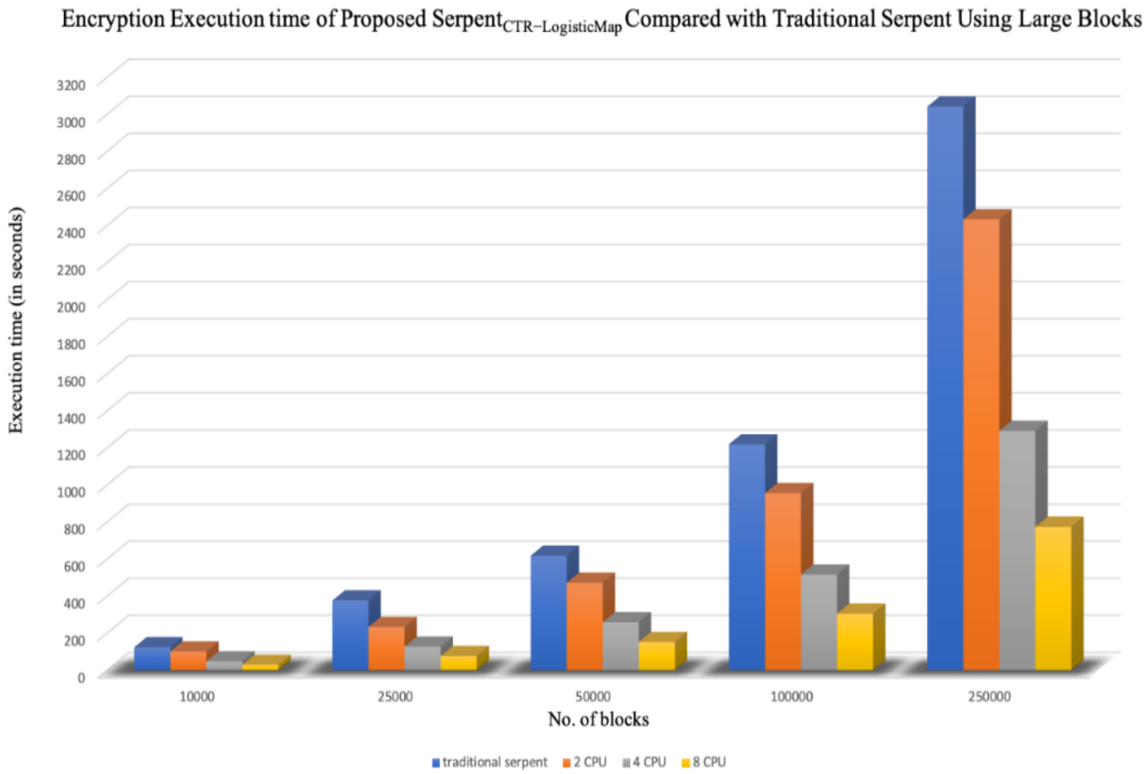
Block Size (in bits)	Encryption execution time (in sec)				Encryption reduction rate (in %)		
	Traditional Serpent	Proposed $\mathbf{S}erpent_{CTR-LogisticMap}$					
		2 CPUs	4 CPUs	8 CPUs	2 CPUs	4 CPUs	8 CPUs
Small Block Sizes							
64	0.88	0.59	0.41	0.34	32.23	54.13	61.42
128	1.53	0.99	0.61	0.47	35.18	60.12	69.27
256	3.18	1.91	1.09	0.77	39.85	65.72	75.65
512	6.35	3.67	1.98	1.36	42.08	68.79	78.53
1024	12.97	7.04	3.75	2.49	45.72	71.06	80.81
Large Block Sizes							
10 000	123.33	101.31	47.89	28.95	17.85	61.17	76.52
25 000	376.24	232.99	126.29	75.98	38.08	66.43	79.81
50 000	617.85	471.48	258.87	152.55	23.69	58.10	75.31
100 000	1218.84	955.74	515.98	305.28	21.58	57.66	74.95
250 000	3040.86	2432.89	1291.02	773.56	19.99	57.54	74.56

Table 2. Decryption execution time and reduction rate (in %) of proposed $Serpent_{CTR-LogisticMap}$ compared with traditional Serpent for different block sizes

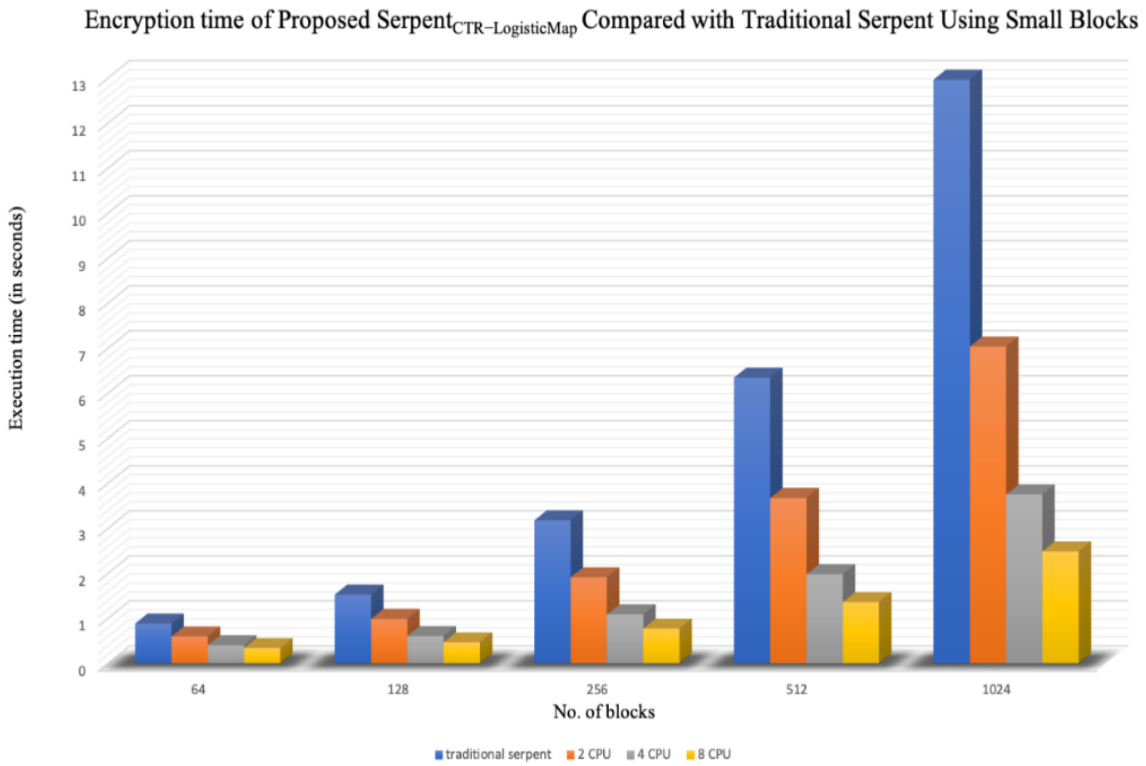
Block size (in bits)	Decryption execution time (in sec)				Decryption reduction rate (in %)		
	Traditional Serpent	Proposed $\mathbf{S}erpent_{CTR-LogisticMap}$					
		2 CPUs	4 CPUs	8 CPUs	2 CPUs	4 CPUs	8 CPUs
Small block sizes							
64	0.74	0.61	0.39	0.32	17.75	46.65	56.55
128	1.47	0.95	0.61	0.46	35.37	58.78	68.43
256	2.98	2.02	1.11	0.76	32.23	62.87	74.33
512	5.89	3.69	1.99	1.33	37.17	66.27	77.45
1024	11.81	6.93	3.77	2.64	41.28	68.10	77.66
Large Block Sizes							
10 000	114.83	92.95	49.49	30.12	19.05	56.90	73.77
25 000	289.62	235.73	128.84	76.15	18.61	55.51	73.71
50 000	565.58	469.95	260.06	153.54	16.91	54.02	72.85
100 000	1136.46	982.27	513.73	306.32	13.57	54.79	73.05
250 000	2829.73	2349.86	1300.52	758.89	16.96	54.04	73.18

4.2.1. The encryption process performance using 2, 4, and 8 CPUs

Looking at Table 1, Figure 2(a) and 2(b) it is very obvious that the proposed $\mathbf{S}erpent_{CTR-LogisticMap}$ method outperformed the traditional Serpent in the encryption process. Regarding small block sizes, it is very blatant that the encryption time is much less especially when considering 8 CPUs. The more the number of CPUs, the less the encryption execution time. The reduction rates for the three tested CPUs are presented in the last three columns of Table 1, and the proposed method achieved up to 77.66% reduction rate for 1,024 bits of data compared with the traditional Serpent. Even for much larger block sizes, the reduction rate was on average 73% when juxtaposed with traditional Serpent. Figures 3(a) and 3(b) depicts the reduction rates for different block sizes (small size and large size), respectively. It is noteworthy to promulgate that the encryption reduction rate for the proposed method parallel execution of small block sizes is certainly less and will increase just as the block size increases. Nonetheless, the increase will remain roughly constant after a particular value as clearly manifested in Table 1.

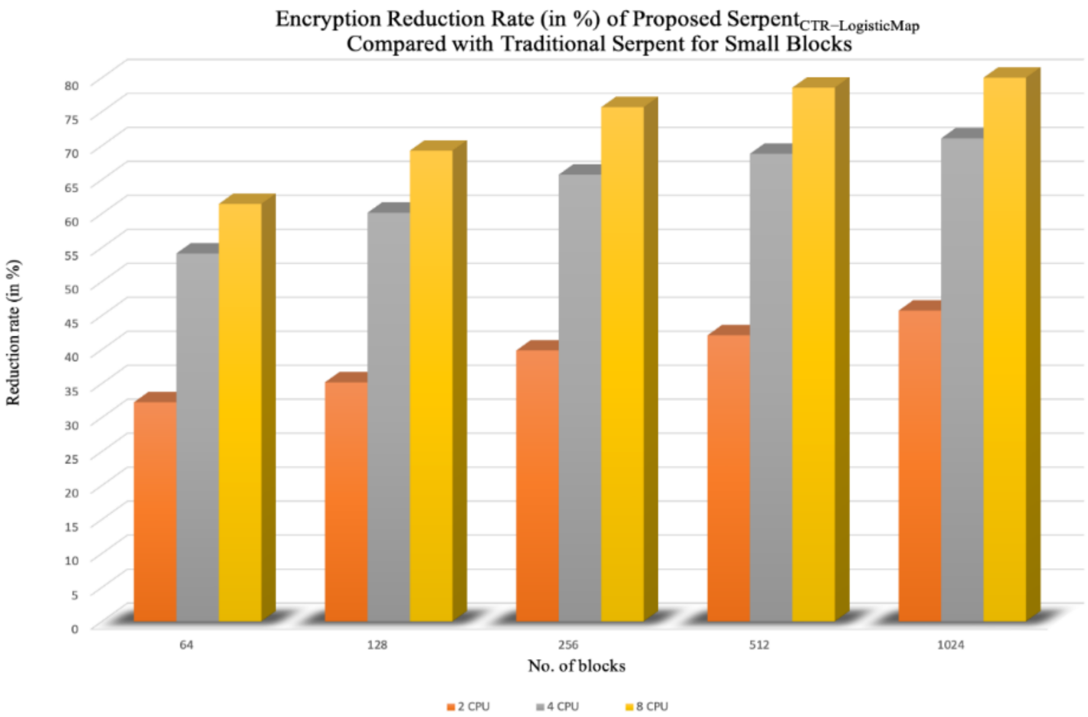


(a)

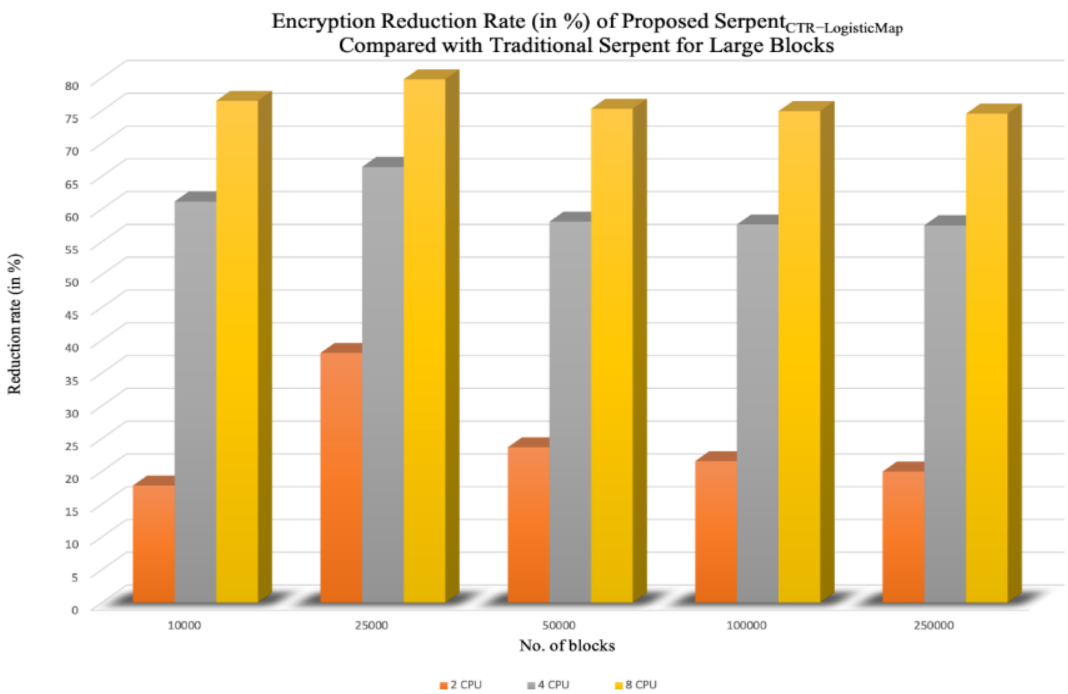


(b)

Figure 2. Encryption execution time of proposed $Serpent_{CTR-LogisticMap}$ compared with traditional Serpent using different block sizes, (a) large size, (b) small size



(a)

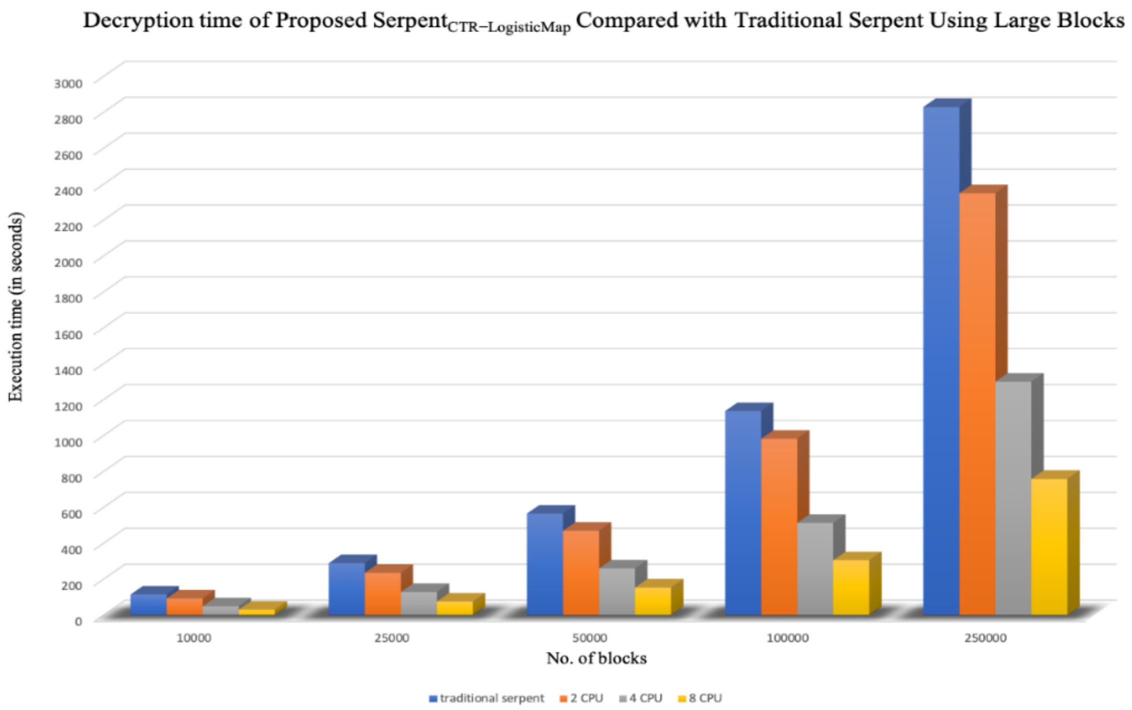


(b)

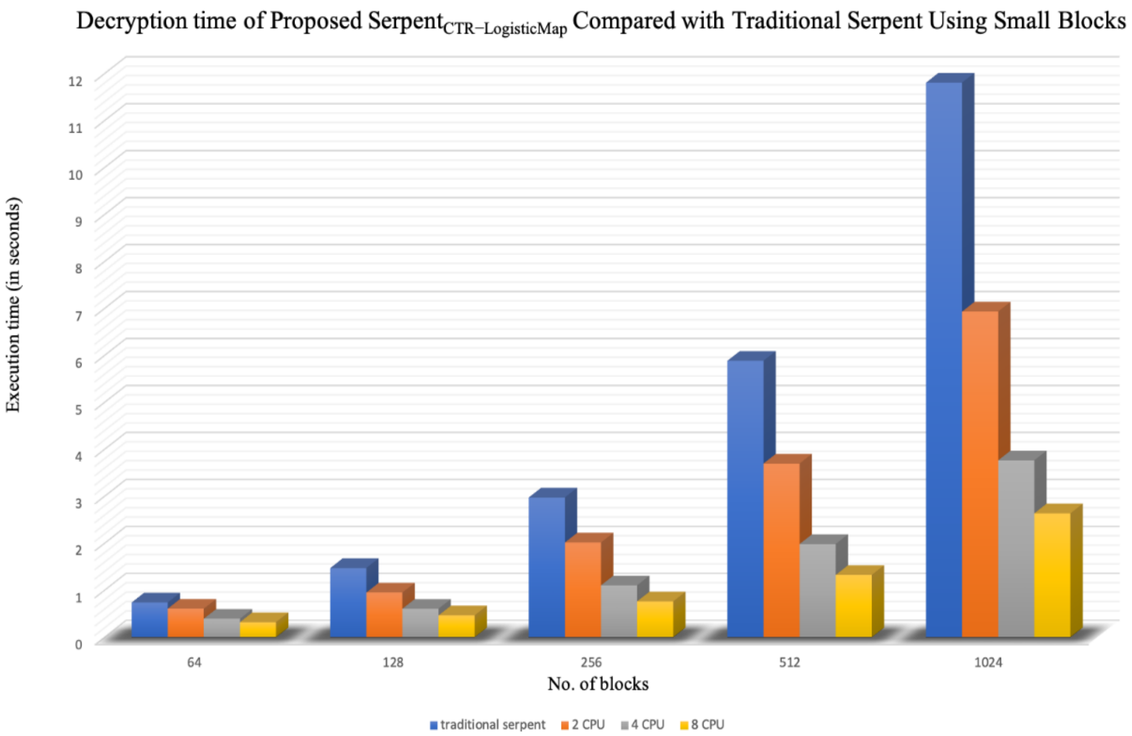
Figure 3. Reduction rate of encryption execution time of proposed $\boldsymbol{Serpent}_{\mathbf{CTR-LogisticMap}}$ compared with traditional Serpent using different block sizes, (a) small size, (b) large size

4.2.2. The decryption process performance using 2, 4 and 8 CPUs

Table 2 and Figures 4(a) and 4(b) manifest the surpass of the proposed $\boldsymbol{Serpent}_{\mathbf{CTR-LogisticMap}}$ method in the decryption execution time. Using 8 CPUs, alleviated the reduction rate up to 77.66% for 1024 bytes of block data and an average of 73.5% for large data blocks. Scrutinizing the last three columns of Table 2, evidently proclaim that the proposed method outcompetes the standard Serpent and lessened the decryption time greatly. This is evinced in Figures 5(a) and 5(b) for small and large data blocks apiece. Consequently, using parallel processing and hence multiple CPUs abate the execution performance to a great extent.

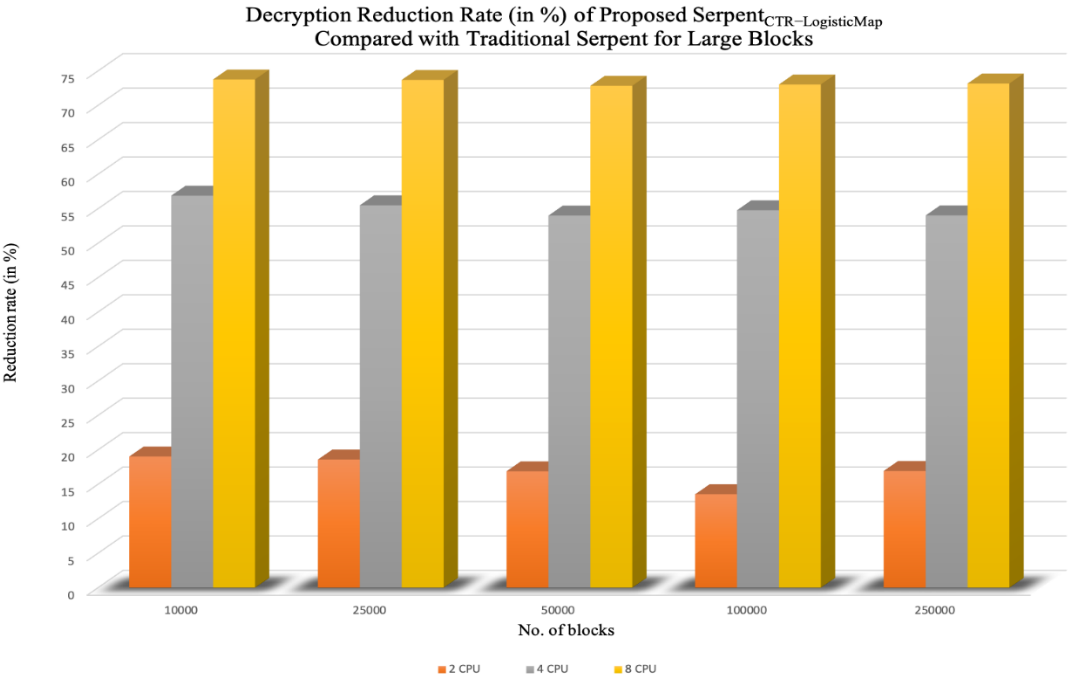


(a)

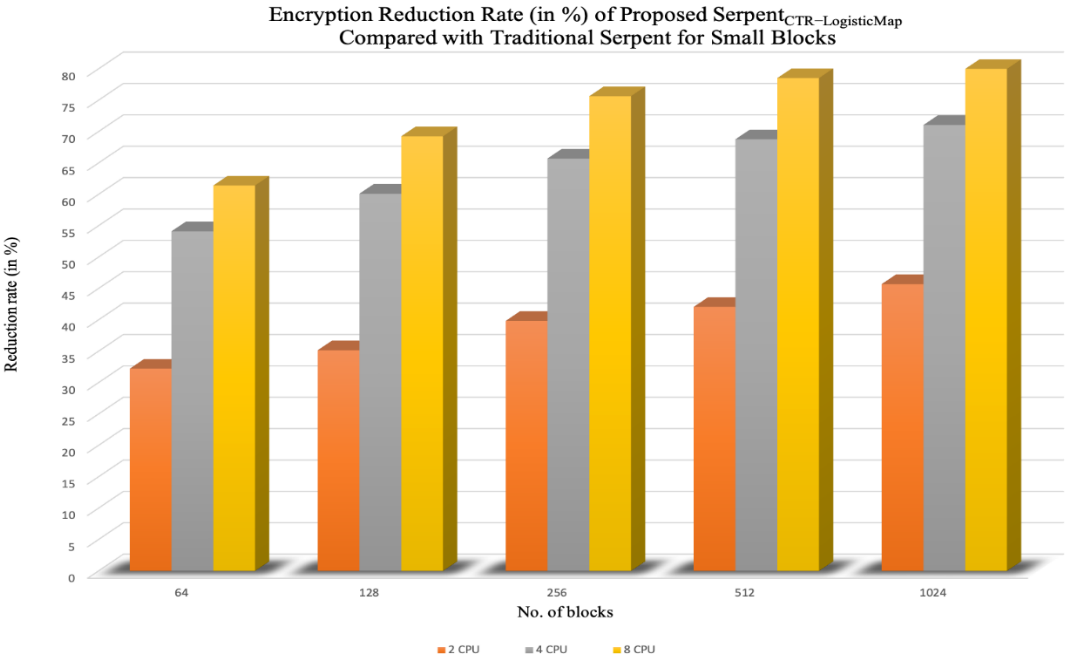


(b)

Figure 4. Decryption execution time of proposed $\boldsymbol{Serpent}_{\mathbf{CTR-LogisticMap}}$ compared with traditional Serpent using different block sizes, (a) large size, (b) small size



(a)

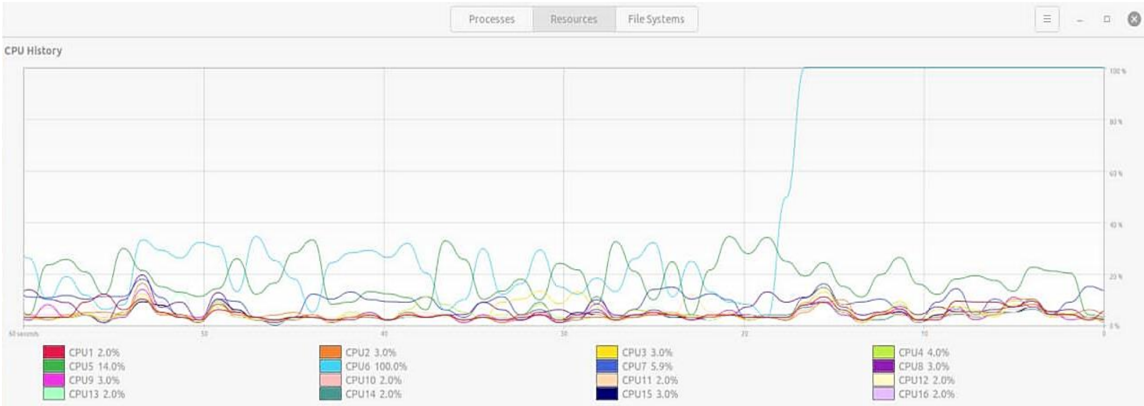


(b)

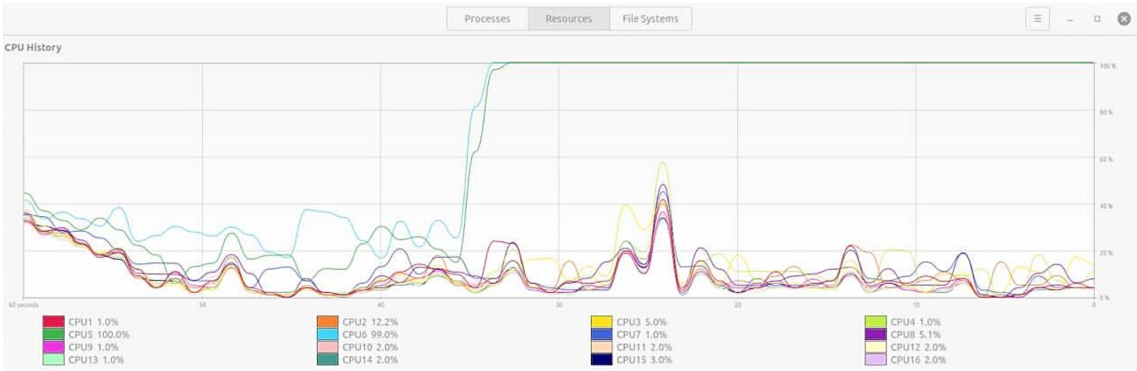
Figure 5. Reduction rate of decryption execution time of proposed $\boldsymbol{S}erpent_{CTR-LogisticMap}$ compared with traditional Serpent using different block sizes, (a) large size, (b) small size

4.2.3. CPU usage analysis

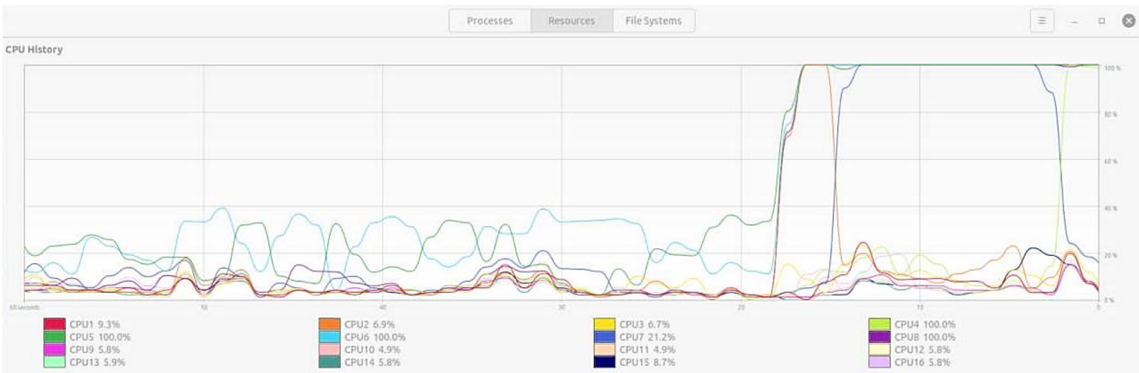
Figure 6 displays the CPU usage of the traditional Serpent (using one CPU) together with that of the proposed $\boldsymbol{Serpent}_{\mathbf{CTR-LogisticMap}}$ method when using 2, 4, and 8 CPUs respectively. In Figure 6(a), only CPU 6 is utilized fully, while Figure 6(b) shows the full usage of both CPU 5 and CPU 6. On the other hand, Figures 6(c) and 6(d) display the exploitation of utterly multiple CPUs and hence the superior utilization of the CPUs. This has a high effect on the CPU usage and clearly the proposed method prevailed over the traditional Serpent with respect to CPU usage.



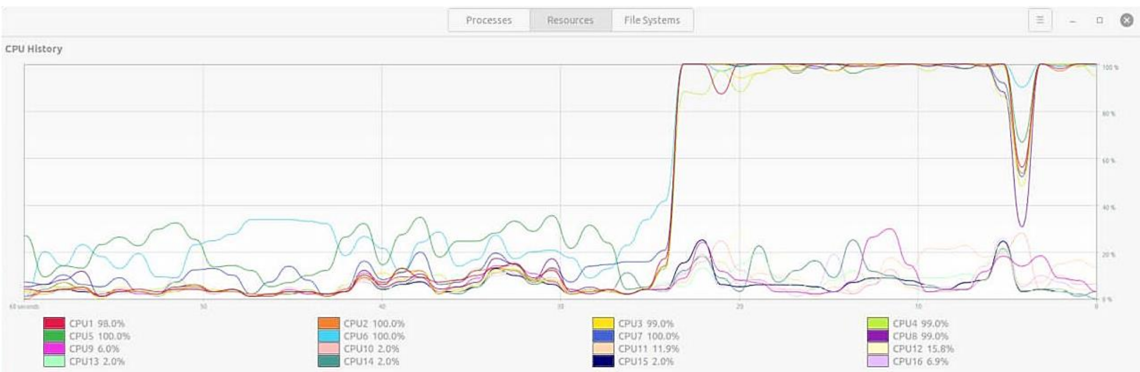
(a)



(b)



(c)



(d)

Figure 6. CPU usage (a) traditional Serpent using ONLY one CPU; (b) $\boldsymbol{Serpent}_{\mathbf{CTR-LogisticMap}}$ using CPU 5 and CPU 6; (c) $\boldsymbol{Serpent}_{\mathbf{CTR-LogisticMap}}$ using CPUs 4, 5, 6, and 8; and (d) $\boldsymbol{Serpent}_{\mathbf{CTR-LogisticMap}}$ using 8 CPUs

4.3. Security analysis
4.3.1. Key space analysis
For an encryption system to be impervious to brute force attack, it is recommended to have a large key space. The input to the proposed method besides the randomly generated 256-bit key is the set IV, Nonce, and ID. These are initial conditions and control parameters to the Logistic Map, which are all double-precision numbers. Therefore, if the computational precision of each of r and x are 10^{-16}, then the key space is greater than $10^{16}\times10^{16}\times$Key. Hence, from the security aspect, the key space of the proposed method is analyzed and is given by (1):

$$n \times 10^{16} \times 10^{16} \times 2^{256} \quad (1)$$

where n is the number of blocks.

The traditional Serpent, on the other hand, has a key space of only 2256. This is incomparable to our proposed method, which is $n \times 10^{16} \times 10^{16} \times 2^{256}$. Furthermore, the more number of processes used, the better execution time as demonstrated in Figures 2 and 4. Hence, our proposed method excels the traditional Serpent in terms of key space.

Ergo, the proposed method has an immense sufficient key space to resist all sorts of brute-force attacks, ensuring its effectiveness. Furthermore, because the proposed method is chaos-based, it is inferred that any small change in the initial conditions or parameter change values, will induce an unsuccessful decryption of the ciphertext [45]. Hence, the precise knowledge of the given key is necessary for decrypting the ciphertext [40].

4.3.2. Strict avalanche criterion
Avalanche criterion is one of the most vital criteria in assessing the strength of any cipher technique. It evaluates how much the ciphertext changes as a result of a small change in the input. It is highly recommendable in any cipher technique to have avalanche score in the range 0.5±ε [46]. That is to say, if only one bit is modified in the input, then every output bit has a probability of 0.5±ε to change. Our proposed method has two inputs, key and plaintext. Therefore, the two aspects of avalanche criterion:

- Key avalanche: changing just one bit in the key and using the same plaintext, the ciphertext will hence change with a probability of 0.5±ε.
- Plaintext avalanche: changing just one bit in the plaintext and using the same key, and there will be a probability change of 0.5±ε.

The avalanche score is gauged as (2).

$$\frac{Key}{plaintext} avalanche\ score = (number\ of\ changed\ bits) \\ / (total\ number\ of\ key/plaintext\ bits) \times 100 \quad (2)$$

Twelve experiments were conducted using different plaintext sizes, consequently the key and plaintext avalanche scores were evaluated and gave favorable results as demonstrated in Table 3. This attested that any incorrect guess will entirely change the obtained ciphertext. Conversely, any mistaken key prediction will produce a completely wrong plaintext. Noteworthy, that when the plaintext bit is changed in the middle, the avalanche score is higher than when the changed bit is at the end. In Table 4, a detailed example is explained showing an input key, 128 bits plaintext and their corresponding generated ciphertext. Then, the second row of the table shows the key avalanche where one bit in the key (shown in red) is changed to produce a ciphertext having 52.87% avalanche score. Similarly, the third row presents the plaintext avalanche score of 50.29% when just one bit in the plaintext (shown in red) was changed. This affirms that the proposed method exhibits avalanche criteria.

Table 3. Key and plaintext avalanche scores for different input plaintext sizes

Number of bits in plain-text/ciphertext	Key Avalanche		Plaintext Avalanche	
	# of changed bits	% of changed bits	# of changed bits	% of changed bits
128	86	50.29%	80	47.62%
128	92	52.87%	86	50.29%
384	196	40.66%	206	42.30%
256	130	40.50%	148	44.85%
512	264	40.93%	304	45.78%
640	340	41.98%	338	41.78%

Table 4. An example for key and plaintext avalanche scores for 128 bits input plaintext

Avalanche Criteria	Input Key	Plaintext 128 bits	Ciphertext 128 bits	Avalanche Score
	0c0c c93b d432	1001 1000 0010 1111	0001 1001 1110 0101	
	fe0e 9a97 6a6b	0000 1011 1001 0100	1000 0000 0011 1001	
	0e8e ac43 a61c	1110 0010 0110 0010	1110 1111 0001 1111	
	0103 259a edb8	0111 0110 0100 1110	1110 0000 0010 1010	
	e2a9 7b64 0817	0000 1011 0011 1100	1000 0100 1111 1110	
		0100 0101 1011 0100	0101 0010 1110 0000	
		1100 0010 1110 1110	0001 1001 0100 1101	
		0010 1111 1101 1101	0000 1011 0111 1011	
Key avalanche	0c0c c93b d432	1001 1000 0010 1111	0011 0000 1011 1100	52.87%
	fe0e 9a97 6a6b	0000 1011 1001 0100	0010 0111 1101 0010	
	0e8e ac43 a61c	1110 0010 0110 0010	0110 1101 0001 1101	
	0103 259a edb8	0111 0110 0100 1110	1101 1001 1110 0011	
	e2a9 7b64 081**5**	0000 1011 0011 1100	1011 0101 0110 0001	
		0100 0101 1011 0100	0101 0100 0101 1110	
		1100 0010 1110 1110	1011 1111 1111 0101	
		0010 1111 1101 1101	1000 1110 1001 1101	
Plaintext avalanche	0c0c c93b d432	1001 1000 0010 1111	1001 0001 1010 0110	50.29%
	fe0e 9a97 6a6b	0000 1011 1001 0100	0001 1100 1001 1001	
	0e8e ac43 a61c	1110 0010 0110 0010	0100 0111 0000 0011	
	0103 259a edb8	0111 0110 0100 111**1**	1010 1100 1001 0001	
	e2a9 7b64 0817	0000 1011 0011 1100	0011 0010 0110 1110	
		0100 0101 1011 0100	0111 0100 0011 0111	
		1100 0010 1110 1110	0001 1010 1101 1101	
		0010 1111 1101 1101	1010 0000 0001 0001	

4.3.3. The security of the logistic map

Due to the chaotic systems' characteristics of being unforeseeable, random, ergodic and high sensitive to preliminary conditions, they are well suited to crypto systems and secure communication [6]. The proposed $\boldsymbol{Serpent}_{\textbf{CTR-LogisticMap}}$ method improves the security by generating distinct block sub-keys utilizing logistic map. Ergo, adding intricacy to the proposed enhanced Serpent. Furthermore, all block sub-keys can be generated prior to the inauguration of the enhanced Serpent, consequently running the blocks in parallel and further conceal the plaintext patterns. Noteworthy to mention that executing the ameliorated Serpent in parallel CTR mode results in expeditious execution.

4.3.4. The randomness of the proposed method $\boldsymbol{Serpent_{CTR-LogisticMap}}$

The statistical test suite (STS) recommended by the NIST [18], [46] was used to evaluate the randomness feature of the proposed $Serpent_{CTR-LogisticMap}$ method. This NIST suite test verifies that the produced output is statistically imperceptible from an unpredictable random output. A probability, P-value, is calculated for 15 different tests, where a range [0.01, 1] for P is required for passing the tests. The experiments have been performed on a significance level of 0.01, which certifies that the likelihood of declining the null hypothesis while it is true is 0.01 [18], [46]. The proposed method succeeded in passing all tests as verified in Figure 7 and therefore signifies its efficiency.

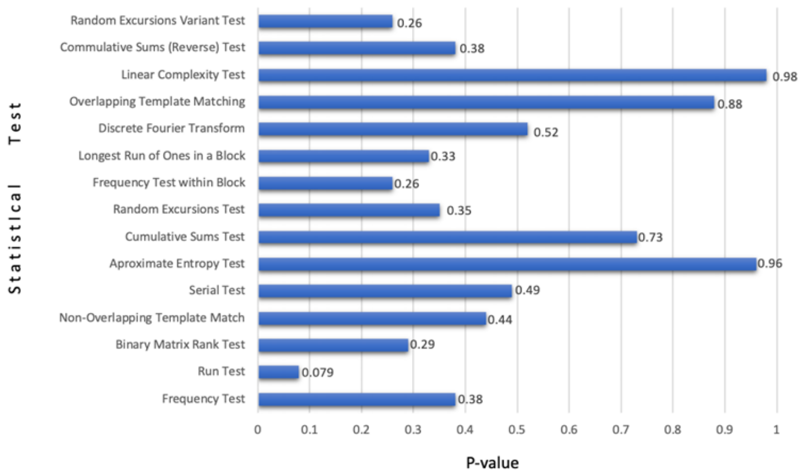


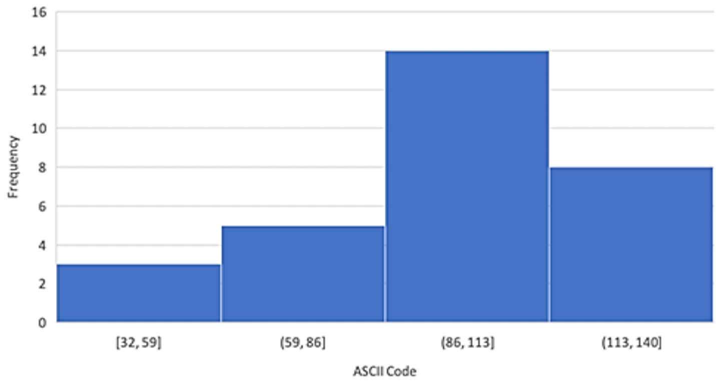
Figure 7. NIST statistical test suite for scrutinizing the proposed $\boldsymbol{Serpent}_{\textbf{CTR-LogisticMap}}$

4.4. The statistical analysis
4.4.1. Histogram analysis

The frequency analysis is based on histograms, which measures if any data can be collected about the plaintext by scrutinizing a histogram [39]. Figure 8 depicts the histogram analysis. The plaintext histogram is shown in Figure 8(a), whilst Figure 8(b) demonstrates the histogram of ciphertext which is uniform and does not testify any information about the plaintext that generated it. Ergo, the proposed $Serpent_{CTR-LogisticMap}$ method is robust against histogram attacks in addition to frequency attacks.

Plaintext (280 characters):

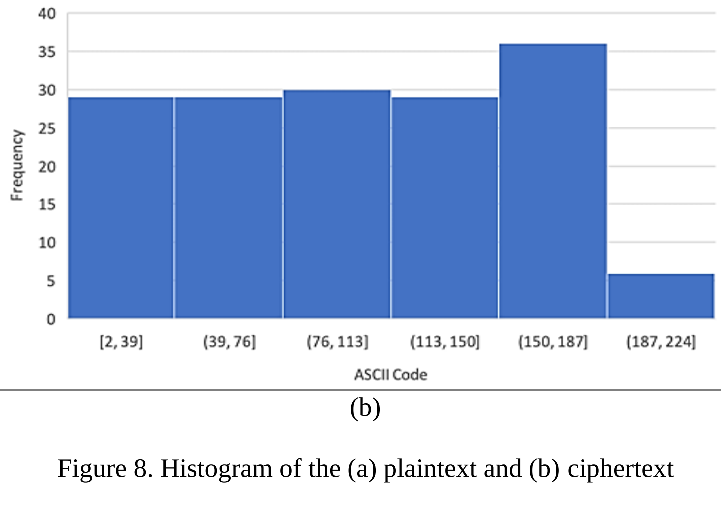
"Once upon a time there was a dear little girl who was loved by every one who looked at her, but most of all by her grandmother, and there was nothing that she would not have given to the child. Once she gave her a little cap of red velvet, which suited her so well that she would never wear anything else. She was always called little red riding hood."



(a)

Ciphertext:

¹Ï/òNw.EU»@;Í\W»2K¨ê@QôÕÌï'4åø,§KÊF¥N´°ÂSSÆî<â`.¥/
8ÿ4m+ÇÕ°ùHõòÈ®M:@XTÝu¤'Úäþs¾atSDÉ"Yúið_# QÖ>KF
¶¥Mó3¿Þð «í¿ûÁù[YÖsÉ@Ô#úª±q6ûFqPÝQIx¹¬Ô
TÚÈB~·N~<ö°þî`Eþ¤TÌ<ïåKx¬.¬ég-%¾¥)ÑÏUzâú)IvÙ ü
´Zä7¾këêDjVwð°e~p6A¼L1£ÂÕ¬Îc ³%½6v¶ÃVúÂÍÊ@"J¤
-Ä¸(p ÌéûüôÈÃBPØ1ǁfÁc&ÃÉÝ÷Ã`3òL¡Õ%æb7Å'~F



(b)

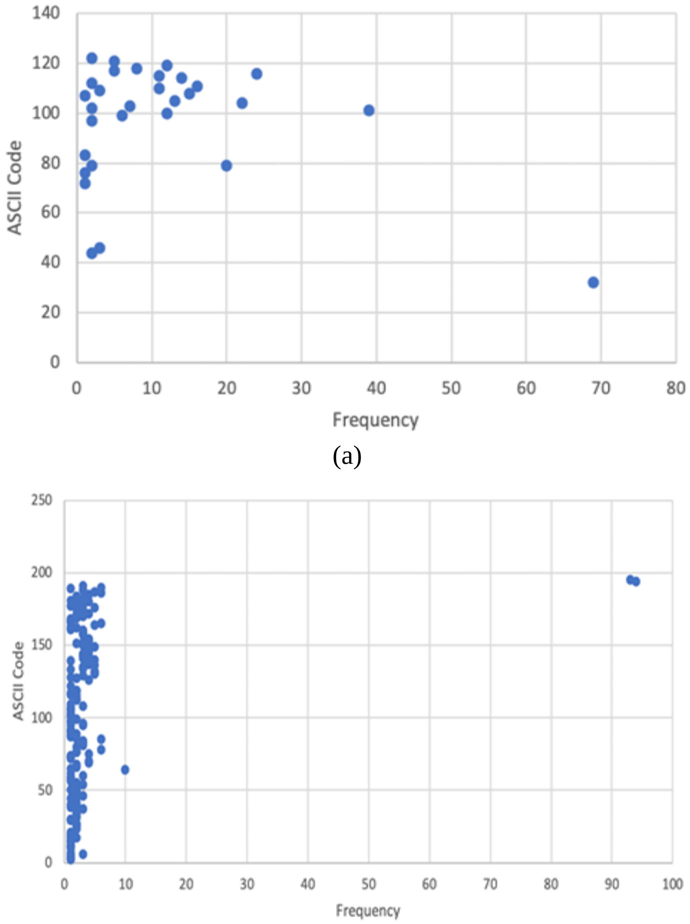
Figure 8. Histogram of the (a) plaintext and (b) ciphertext

4.4.2. Correlation coefficient analysis

To avoid statistical attacks, correlation coefficient analysis is used to measure the association between the plaintext and ciphertext [39], [47]. It is calculated using (3). Figures 9(a) and 9(b) depict the correlation distribution of adjacent bits in a 280-characters plaintext and ciphertext respectively. It is evident Figure 9(b) that the correlation distribution of ciphertext is uniform when juxtaposed with the plaintext generating it. The attained correlation values for plaintext and ciphertext were very close to zero, having values of -0.2709 and 0.2341 respectively. Table 5 shows the correlation results when considering different text files of different sizes. These robust confounding properties achieved by the examined correlation analysis evade statistical attacks and affirms the efficacy of the proposed method.

$$Correlation\ cofficient\ = \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum(x_i-\bar{x})^2\Sigma(y_i-\bar{y})^2}} \quad (3)$$

where x_i and y_i are the values of the x and y variables in a sample, and $(\bar{x})$ and $(\bar{y})$ are the mean values of the x and y variables respectively.



(b)

Figure 9. Correlation analysis of (a) the plaintext (280 characters) and (b) the ciphertext

Table 5. The results of correlation between different text files

File name	Correlation	Size (in Bytes)
Plaintext1 & ciphertext1	0.039694093	256
Plaintext2 & ciphertext2	-0.2709	280
Plaintext3 & ciphertext3	0.154811841	352
Plaintext4 & ciphertext4	-0.021667779	496

4.5. Comparison with related schemes

This section shed the light on the propitious prospective of the proposed method by comparing it with related schemes, namely Traditional Serpent, AES running in multiprocessing environment and references [1], [36]–[41], [43], [44], [48]–[54]. It is juxtaposed in terms of performance, security analysis including key space, key sensitivity, plaintext avalanche criterion, in addition to statistical analysis, namely histogram analysis and correlation coefficient analysis.

4.5.1. Comparing the time execution performance

Table 6 presents the execution time taken by the proposed $\boldsymbol{S}erpent_{CTR-LogisticMap}$ method juxtaposed with Pendli *et al.* [1], Shah *et al.* [36], Zagi and Maolood [41], Shah *et al.* [48], Elkamchouchi *et al.* [37] and Elshoush *et al.* [43]. It is very obvious that the proposed method is superior to the current methods manifesting an encryption reduction of up to 80.81% compared with the maximum procured by Shah *et al.* [48] attaining an encryption reduction rate of 52.05%.

Table 6. Comparison of the time execution reduction rate in % for the proposed method with different Serpent enhanced schemes and a multiprocessor AES

Text encryption algorithm	Reference	Encryption reduction rate (in %)	Decryption reduction rate (in %)
Multiprocessing AES	Pendli *et al.* [1] 2016	40-45%	38-45%
Serpent-based	Shah *et al.* [36] 2018	16.54%	30.11%
	Zagi *et al.* [41] 2020	20.63%-23.8%	27.08% -38.54%
	Shah *et al.* [48] 2020	52.05%	52.31%
Serpent and Chaotic-based	Elkamchouchi *et al.* [37] 2018	50.3%	51.87%
	Elshoush *et al.* [43] 2022	48.1%-53.2%	45.3%-51.5%
	Proposed $\boldsymbol{S}erpent_{CTR-LogisticMap}$	61.42%- 80.81%	56.55%-77.66%

4.5.2. Comparison of key space analysis

It is essential for every cryptosystem to have a large key space to be infallible versus a brute-force attack. Table 7 shows the key space of our proposed enhanced Serpent compared to different encryption schemes, specifically serpent-based, chaotic-based and Serpent-chaotic-based ones. Blatantly, the proposed $Serpent_{CTR-LogisticMap}$ outperforms all related schemes and doubtlessly substantiated its robustness and resistance to brute force attack.

Table **7**. Comparison of key space for the proposed method and related encryption schemes

Text encryption algorithm	Reference	Key space
Serpent-based	Traditional Serpent	2^{256}
	Tayel *et al.* [50] 2018	2^{256}
	Shah *et al.* [48] 2020	2^{264}
	Hussain *et al.* [44] 2023	2^{100}
Chaotic-based	Murillo-Escobar *et al.* [52] 2014	2^{128}
	Ekhlas *et al.* [39] 2017	$2^{213}((10^{16})^4)$
	Menon *et al.* [51] 2020	2^{128}
	OleiwiTuama *et al.* [54] 2021	$2^{545}(10^{15})^{11}$
	Charalampidis *et al.* [40] 2022	2^{480}
Serpent and Chaotic-based	Elkamchouchi *et al.* [37] 2018	128^{10}
	Yousif [38] 2019	10^{112}
	Elshoush *et al.* [43] 2022	$n\times2^{256}$ (n=no. of blocks)
	Proposed $\boldsymbol{S}erpent_{CTR-LogisticMap}$	$n\times10^{16}\times10^{16}\times2^{256}$ (n=no. of blocks)

4.5.3. Comparison of key sensitivity

A good cryptosystem must be highly sensitive to secret keys which reveals how ciphertext alters as a result of a tiny change in the secret key. Section 4.3.2. proffer how the proposed system unveils key sensitivity with a score of 50.29%. Even when compared with related works of Elkamchouchi *et al.* [37], Ekhlas *et al.* [39], Murillo-Escobar *et al.* [52], Mangi *et al.* [53] and OleiwiTuama *et al.* [54], our proposed method blatantly achieved superb results compared to their claimed results.

4.5.4. Comparison of plaintext avalanche effect

Table 8 presents the plaintext avalanche of the proposed method together with recent work. Our proposed method achieved supreme results for plaintext avalanche as elucidated in section 4.3.2. when juxtaposed with the work of Hussain *et al.* [44] and Mangi *et al.* [53] as delineated in Table 8. This affirms the efficacy of the proposed method in terms of plaintext avalanche effect.

Table 8. Comparison of plaintext avalanche for the proposed method and related work of [44] and [53]

Text encryption algorithm	Reference	Plaintext avalanche
Serpent-based	Hussain *et al.* [44]	50.0%
Logistic map	Mangi *et al.* [53]	52.04%
Serpent and Chaotic-based	Proposed $\boldsymbol{S}erpent_{CTR-LogisticMap}$	52.87%

4.5.5. Comparison of histogram analysis

Figure 8 shows the histograms of our proposed method. Obviously, they show uniformity of ciphertext histograms. Thus referring to the histograms of researches Elkamchouchi *et al.* [37], Ekhlas *et al.* [39], Murillo-Escobar *et al.* [52], Charalampidis *et al.* [40] and OleiwiTuama *et al.* [54] our proposed method's results were superlative showing uniformity of ciphertext histogram.

4.5.6. Comparison of correlation coefficient

Regarding correlations coefficient, Table 9 presents results achieved by different researches. The proposed method attained good results -0.2709. Palpably our proposed method attained excellent results compared to prevailing schemes.

Table 9. Comparison of correlation coefficient for the proposed method and related encryption schemes

Text encryption algorithm	Reference	Correlation coefficient
Serpent-based	Traditional Serpent	0.0814
Chaotic-based	Mangi *et al.* [53]	-0.0024
	Ekhlas *et al.*[39]	-0.0215
Serpent and Chaotic-based	Ali *et al.* [49]	0.0023
	Elkamchouchi *et al.* [37]	0.006
	Proposed $\boldsymbol{S}erpent_{CTR-LogisticMap}$	-0.2709

4. CONCLUSION

Yet the most secure and robust cipher will be deemed impractical if it has substandard performance. In this paper, we proffered an enhanced $\boldsymbol{S}erpent_{CTR-LogisticMap}$ for encrypting text using logistic map and running in multiprocessing CTR mode. The security is further boosted by randomly generating intricate sub-keys for each block using logistic map. Using Python 3.9, we comprehensively tested the enhanced $\boldsymbol{S}erpent_{CTR-LogisticMap}$ using the most widely metrics, viz. performance execution time, CPU usage, key space analysis, key/plaintext avalanche effect, histogram analysis and correlation coefficient analysis. Compared to the traditional Serpent, it effectively achieved an encryption reduction rate of up to 80.81% for 1K block and 74.56% for 250 000 bits blocks of data. Whereas the decryption reduction rate came up to 77.66% for 1K block and 73.18% for 250 000 bits data blocks. It is noteworthy to mention that the reduction rate increased considerably when encrypting/decrypting small texts whilst stayed on an average reduction rate when applied to large text. It has an excessive key space due its dependency on the number of blocks and the randomly generated logistic map keys, thus giving it superiority regarding brute force attacks. Moreover, any change in key/plaintext made a considerable percentage change in the ciphertext, thus exhibiting avalanche criteria, specifically 52.87% for key avalanche and 50.29% for plaintext avalanche for 128 bits of plaintext. Additionally, it withstood statistical and frequency attacks. Furthermore, NIST Statistical Test Suite (STS) was utilized to test its randomness, where it passed all tests successfully. When juxtaposed with prevailing methods, the expeditious proposed method was infallible and beyond comparison and assuredly confirmed its efficacy. As future work, we recommend testing the proposed method in encrypting/decrypting images, seeing that it is unquestionably will attain superior results.

DATA AVILIBILTY

The data is available at GitHub: Enhanced $Serpent_{CTR-LogisticMap}$ Code.

REFERENCES

[1] V. Pendli, M. Pathuri, S. Yandrathi, and A. Razaque, "Improvising performance of advanced encryption standard algorithm," *2016 Second International Conference on Mobile and Secure Services (MobiSecServ)*, Gainesville, FL, USA, 2016, pp. 1-5, doi: 10.1109/MOBISECSERV.2016.7440224.

[2] M. E. Smid, "Development of the advanced encryption standard," *Journal of Research of the National Institute of Standards and Technology*, vol. 126, p. 126024, Aug. 2021, doi: 10.6028/jres.126.024.

[3] S. Yadav, U. Verma, and C. Bhardwaj, "Data security in cloud computing using homomorphic encryption," *International Journal of Scientific Research*, vol. 3, no. 5, pp. 78–81, Jun. 2012, doi: 10.15373/22778179/may2014/26.

[4] R. Anderson, "Serpent - A candidate block cipher for the advanced encryption standard," *University of Cambridge*, 2001.

[5] M. Nagendra and M. C. Sekhar, "Performance improvement of advanced encryption algorithm using parallel computation," *International Journal of Software Engineering and its Applications*, vol. 8, no. 2, pp. 287–296, 2014, doi: 10.14257/ijseia.2014.8.2.28.

[6] H. T. Elshoush, B. M. Al-Tayeb, and K. T. Obeid, "Enhanced serpent algorithm using Lorenz 96 chaos-based block key generation and parallel computing for RGB image encryption," *PeerJ Computer Science*, vol. 7, p. e812, Dec. 2021, doi: 10.7717/PEERJ-CS.812.

[7] B. Najafi, B. Sadeghian, M. S. Zamani, and A. Valizadeh, "High speed implementation of serpent algorithm," in *Proceedings of the International Conference on Microelectronics, ICM*, 2004, pp. 718–721, doi: 10.1109/icm.2004.1434767.

[8] K. Kabilan, M. Saketh, and K. K. Nagarajan, “Implementation of Serpent cryptographic algorithm for secured data transmission,” in *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Mar. 2017, pp. 1–6, doi: 10.1109/ICIIECS.2017.8275863.
[9] R. Anderson, E. Biham, and L. Knudsen, “The case for serpent,” *AES Candidate Conference*, pp. 349–353, 2000.
[10] S. Simha, Prathibha, and H. Priya, “Enhancing cloud security with the implementation of serpent encryption algorithm,” *Imperial Journal of Interdisciplinary Research*, vol. 3, no. 5, 2017.
[11] D. A. Osvik, “Speeding up serpent,” in *AES candidate conference*, 2000, pp. 317–329.
[12] M. H. Taher, A. E. T. El_Deen, and M. E.Abo-Elsoud, “Hardware implementation of the serpent block cipher using FPGA technology,” *International Journal of Electronics and Communication Engineering & Technology (IJECET)*, vol. 5, no. 10, pp. 34–44, 2014.
[13] D. Ivančić, D. Runje, and M. Kovač, “Implementation of serpent encryption algorithm on 24-bit DSP processor,” in *International Symposium on Image and Signal Processing and Analysis, ISPA*, 2001, pp. 411–416, doi: 10.1109/ISPA.2001.938665.
[14] M. Naeemabadi, B. S. Ordoubadi, A. M. Dehnavi, and K. Bahaadinbeigy, “Comparison of Serpent, Twofish and Rijndael encryption algorithms in teleophthalmology system,” *Advances in Natural and Applied Sciences*, vol. 9, no. 4, pp. 137–149, 2015.
[15] K. J. Compton, B. Timm, and J. Vanlaven, “A simple power analysis attack on the serpent key schedule,” *Ratio*, pp. 1–10, 2009.
[16] E. Biham, R. Anderson, and L. Knudsen, “Serpent: A new block cipher proposal,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1372, Springer Berlin Heidelberg, 1998, pp. 222–238.
[17] M. A. Al-Shabi, “A survey on symmetric and asymmetric cryptography algorithms in information security,” *International Journal of Scientific and Research Publications (IJSRP)*, vol. 9, no. 3, Mar. 2019, doi: 10.29322/ijsrp.9.03.2019.p8779.
[18] H. T. Elshoush, B. H. Abdallah, D. M. Ahmed, A. A. Ishag, and S. O. Afifi, “Expeditiousness serpent using CTR mode and logistic map,” *Journal of Information Hiding and Multimedia Signal Processing*, vol. 13, no. 3, pp. 214–225, 2022.
[19] M. Aljohani, I. Ahmad, M. Basheri, and M. O. Alassafi, “Performance analysis of cryptographic pseudorandom number generators,” *IEEE Access*, vol. 7, pp. 39794–39805, 2019, doi: 10.1109/ACCESS.2019.2907079.
[20] R. Matthews, “On the derivation of a ‘chaotic’ encryption algorithm,” *Cryptologia*, vol. 13, no. 1, pp. 29–42, Jan. 1989, doi: 10.1080/0161-118991863745.
[21] Z. Lin, G. Wang, X. Wang, S. Yu, and J. Lü, “Security performance analysis of a chaotic stream cipher,” *Nonlinear Dynamics*, vol. 94, no. 2, pp. 1003–1017, Jun. 2018, doi: 10.1007/s11071-018-4406-8.
[22] S. M. H. Alwahbani and H. T. I. Elshoush, “Chaos-based audio steganography and cryptography using LSB method and one-time pad,” in *Lecture Notes in Networks and Systems*, vol. 16, Springer International Publishing, 2018, pp. 755–768.
[23] S. M. H. Alwahbani and H. T. I. Elshoush, “Hybrid audio steganography and cryptography method based on high least significant bit (LSB) layers and one-time pad—A novel approach,” in *Studies in Computational Intelligence*, vol. 751, Springer International Publishing, 2018, pp. 431–453.
[24] K. Audhkhasi, “Chaos based cryptography,” *Citeseerx*, pp. 1–8, 2009, Accessed: Mar. 04, 2024. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=6a8a9ad63e38a8f26f89c56e2d6affc8b6daa584
[25] L. Kocarev, “Chaos-based cryptography: a brief overview,” *IEEE Circuits and Systems Magazine*, vol. 1, no. 3, pp. 6–21, 2001, doi: 10.1109/7384.963463.
[26] D. Xiao, X. Liao, and S. Deng, “One-way hash function construction based on the chaotic map with changeable-parameter,” *Chaos, Solitons and Fractals*, vol. 24, no. 1, pp. 65–71, Apr. 2005, doi: 10.1016/S0960-0779(04)00456-4.
[27] A. G. Marco, A. S. Martinez, and O. M. Bruno, “Fast, parallel and secure cryptography algorithm using Lorenz’s attractor,” *International Journal of Modern Physics C*, vol. 21, no. 3, pp. 365–382, Mar. 2010, doi: 10.1142/S0129183110015166.
[28] O. M. Al-Hazaimeh, M. F. Al-Jamal, N. Alhindawi, and A. Omari, “Image encryption algorithm based on Lorenz chaotic map with dynamic secret keys,” *Neural Computing and Applications*, vol. 31, no. 7, pp. 2395–2405, Aug. 2019, doi: 10.1007/s00521-017-3195-1.
[29] S. C. Phatak and S. S. Rao, “Logistic map: a possible random-number generator,” *Physical Review E*, vol. 51, no. 4, pp. 3670–3678, Apr. 1995, doi: 10.1103/PhysRevE.51.3670.
[30] National Bureau of Standards, “DES modes of operation,” Federal Information Processing Standards Publications (FIPS PUBS), 1980.
[31] M. Dworkin, *Recommendation for block cipher modes of operation methods and techniques*, no. December. 2001.
[32] A. Altigani, M. Abdelmagid, and B. Barry, “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*, vol. 9, no. 28, Jul. 2016, doi: 10.17485/ijst/2016/v9i28/97795.
[33] D. Bujari and E. Aribas, “Comparative analysis of block cipher modes of operation,” *International Advanced Researches and Engineering Congress*, pp. 2–5, 2017.
[34] H. Lipmaa, P. Rogaway, and D. Wagner, “Comments to NIST concerning AES-modes of operations: CTR-mode encryption,” *ResearchGate (unpublished)*, pp. 1–4, 2000.
[35] H. T. Elshoush, B. M. Al-Tayeb, and K. T. Obeid, “New approach for serpent block cipher algorithm based on multi techniques,” *PeerJ Computer Science*, p. 1, 2017, doi: 10.34279/0923-007-003-004.
[36] T. Shah, T. U. Haq, and G. Farooq, “Serpent algorithm: An improvement by 4×4 Sbox from finite chain ring,” in *International Conference on Applied and Engineering Mathematics, Proceedings*, Sep. 2018, pp. 32–37, doi: 10.1109/ICAEM.2018.8536293.
[37] H. M. Elkamchouchi, A. E. Takieldeen, and M. A. Shawky, “A modified serpent based algorithm for image encryption,” in *National Radio Science Conference, NRSC, Proceedings*, Mar. 2018, pp. 239–248, doi: 10.1109/NRSC.2018.8354369.
[38] I. A. Yousif, “Proposed a permutation and substitution methods of serpent block cipher,” *Ibn AL-Haitham Journal for Pure and Applied Sciences*, vol. 32, no. 2, pp. 131–144, May 2019, doi: 10.30526/32.2.2120.
[39] A. Ekhlas, T. Albahrani, and A. Karam, “A text encryption algorithm based on self-synchronizing stream cipher and chaotic maps,” *International Journal of Scientific Research in Science, Engineering and Technology*, 2017, doi: 10.13140/RG.2.2.31642.70085.
[40] N. Charalampidis, C. Volos, L. Moysis, A. V Tutueva, D. Butusov, and I. Stouboulos, “Text encryption based on a novel one dimensional piecewise chaotic map,” in *Proceedings of the 2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering, ElConRus 2022*, Jan. 2022, pp. 263–268, doi: 10.1109/ElConRus54750.2022.9755622.
[41] H. R. Zagi and A. T. Maolood, “A novel serpent algorithm improvement by the key schedule increase security,” *Tikrit Journal of Pure Science*, vol. 25, no. 6, pp. 114–125, Dec. 2020, doi: 10.25130/tjps.v25i6.320.
[42] H. Singh, “Enhancing AES using novel block key generation algorithm and key dependent S-boxes,” *International Journal of Cyber-Security and Digital Forensics*, vol. 5, no. 1, pp. 30–45, 2016, doi: 10.17781/p001985.

[43] H. T. Elshoush, K. T. Obeid, and M. M. Mahmoud, "A novel approach to improve the performance of serpent algorithm using Lorenz 96 chaos-based block key generation," *Journal of Information Hiding and Multimedia Signal Processing*, vol. 13, no. 1, pp. 49–63, 2022.
[44] S. Hussain, M. Asif, T. Shah, A. Mahboob, and S. M. Eldin, "Redesigning the serpent algorithm by PA-loop and its image encryption application," *IEEE Access*, vol. 11, pp. 29698–29710, 2023, doi: 10.1109/ACCESS.2023.3261568.
[45] M. A. Murillo-Escobar, C. Cruz-Hernández, F. Abundiz-Pérez, and R. M. López-Gutiérrez, "Implementation of an improved chaotic encryption algorithm for real-time embedded systems by using a 32-bit microcontroller," *Microprocessors and Microsystems*, vol. 45, pp. 297–309, Sep. 2016, doi: 10.1016/j.micpro.2016.06.004.
[46] M. M. Mahmoud and H. T. Elshoush, "Enhancing LSB using binary message size encoding for high capacity, transparent and secure audio steganography-an innovative approach," *IEEE Access*, vol. 10, pp. 29954–29971, 2022, doi: 10.1109/ACCESS.2022.3155146.
[47] F. Thabit, S. Alhomdy, and S. Jagtap, "Security analysis and performance evaluation of a new lightweight cryptographic algorithm for cloud computing," *Global Transitions Proceedings*, vol. 2, no. 1, pp. 100–110, Jun. 2021, doi: 10.1016/j.gltp.2021.01.014.
[48] T. Shah, T. U. Haq, and G. Farooq, "Improved serpent algorithm: design to RGB image encryption implementation," *IEEE Access*, vol. 8, pp. 52609–52621, 2020, doi: 10.1109/ACCESS.2020.2978083.
[49] Y. Hussain Ali and H. Aabdali Ressan, "Image encryption using block cipher based serpent algorithm," *Engineering and Technology Journal*, vol. 34, no. 2, pp. 278–286, Feb. 2016, doi: 10.30684/etj.34.2b.10.
[50] M. Tayel, G. Dawood, and H. Shawky, "A proposed serpent-elliptic hybrid cryptosystem for multimedia protection," in *2018 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2018*, Sep. 2018, pp. 387–391, doi: 10.1109/ICACCI.2018.8554950.
[51] U. Menon, A. Hudlikar, and A. R. Menon, "A novel chaotic system for text encryption optimized with genetic algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 10, pp. 34–40, 2020, doi: 10.14569/IJACSA.2020.0111005.
[52] M. A. Murillo-Escobar, F. Abundiz-Perez, C. Cruz-Hernandez, and R. M. Lopez-Gutierrez, "A novel symmetric text encryption algorithm based on logistic map," in *the 2014 International Conference on Communications, Signal Processing and Computers (ICNC'14)*, 2014, vol. 2, no. 1, pp. 49–53.
[53] H. T. Mangi, S. A. Ali, and M. J. Jawad, "Encrypting of Text Based on Chaotic Map," *Journal of University of Babylon for Pure and Applied Sciences*, pp. 25–39, Apr. 2023, doi: 10.29196/jubpas.v31i1.4526.
[54] S. OleiwiTuama, S. A. Kadum, and Z. Hussein, "Text encryption approach using DNA computation and hyperchaotic system," in *Proceedings of 2021 2nd Information Technology to Enhance E-Learning and other Application Conference, IT-ELA 2021*, Dec. 2021, pp. 100–105, doi: 10.1109/IT-ELA52201.2021.9773674.

BIOGRAPHIES OF AUTHORS

Huwaida T. Elshoush received the bachelor's degree in computer science (division 1), the master's degree in computer science, and the Ph.D. degree in information security from the Faculty of Mathematical Sciences and Informatics, University of Khartoum, Sudan, in 1994, 2001, and 2012, respectively. Her M.Sc. dissertation dealt with frame relay security. She is currently an associate professor within the Computer Science Department, Faculty of Mathematical Sciences and informatics, University of Khartoum, where she is also acting as the head of research office. Also, she is the deputy dean in the Graduate College at University of Khartoum, Sudan. She has more than 32 publications and some of her publications appeared in Applied Soft Computing Journal (Elsevier), PLOS One Journal, IEEE Access, Multimedia Tools and Applications, PeerJ Computer Science, Journal of Information Hiding and Multimedia Signal Processing and Springer book chapters. Her research interests include the information security, cryptography, steganography, and intrusion detection systems. She is a reviewer of many international reputable journals related to her fields, including Applied Soft Computing Journal (Elsevier). Dr. Elshoush's awards and honors include the second-place prize in the ACM Student Research Com- petition SRC-SAC in 2013 in Coimbra, Portugal. Her article entitled "An improved framework for intrusion alert correlation" has been awarded the Best Student Paper Award of the 2012 International Conference of Information Security and Internet Engineering (ICISIE) in WCE 2012. Other prizes were the best student during the five years of her undergraduate study. She can be contacted at email: htelshoush@uofk.edu.

Duaa M. Ahmed received her bachelor's degree from the Department of Mathematical and Computer Sciences, Faculty of Mathematical Sciences and Informatics, Khartoum University, Sudan, Khartoum. She is currently working with Mobile Telecommunication Network (MTN) Sudan in information Security Department with a bachelor's degree in mathematical and computer sciences. She has one publication in Journal of Information Hiding and Multimedia Signal Processing. She can be contacted at email: duaamusaab27@gmail.com.

Abdalmajid A. Ishag received a B.Sc. degree in computer science from the University of Khartoum, Sudan in 2015, and an MSc degree in computer science which was focused on software engineering in 2018. He is currently working as a software developer at Universal Postal Union (UPU), Bern, Switzerland. He worked on multiple domains using his knowledge of software engineering helping in the process of converting theoretical ideas to a living product. He worked on IoT solutions while he was an IoT Senior developer at Vision Valley (Sudan), and e-government solutions while he was a software engineer at Nile Center for Technology Research (NCTR) (Sudan). In his M.Sc. research in face detection using deep learning, he used his experience to implement his idea, since he decided to participate in the research community by converting theoretical ideas into a working product. He has one publication in Journal of Information Hiding and Multimedia Signal Processing. He can be contacted via email: majid.fms010@gmail.com.

Muawia A. Elsadig received the bachelor's degree in computer engineering, the M.Sc. degree in computer networks, and Ph.D. degree in computer science (information security). He is currently an assistant professor of cybersecurity at the Imam Abdulrahman bin Faisal University (IAU), Dammam, Saudi Arabia. He worked for different accredited international universities and had a rich record of publications in recognized international journals and conferences. He has many years of teaching experience and considerable industry contributions. He has contributed as a reviewer to many reputable international journals and has received many awards for his research activities. His research interests include information security, network security, cybersecurity, wireless sensor networks, bioinformatics, and information extraction. Ranging from theory to design to implementation. He can be contacted at email: muawiasadig@yahoo.com.

Abdelrahman Altigani earned his Ph.D. from the University of Technology, Malaysia. He has published 18 papers in international refereed journals and conferences. He was awarded the best research paper in the computing track at the ICCEEE2013 conference. Additionally, he was a three-time recipient of the International Doctoral Fellowship Scholarship, recognizing his exceptional academic achievements. He has also contributed as a reviewer for numerous research articles in various journals. He can be contacted at email: a.altigani@gmail.com.