Journal of Soft Computing Paradigm (ISSN: 2582-2640)
www.irojournals.com/jscp/

Analysis and Implementation of 3-D Transpositional Cipher Algorithm Based on Rubik's Cube Mechanism

V. Vilasini[1], V. Vasudevan[2], S. Manuj Nanthan[3], K. Lakshmi Kalpana Roy[4]

[1,4]Professor, Computer Science and Engineering, PSG Institute of Technology and Applied Research, Coimbatore, India
[2,3]Computer Science and Engineering, PSG Institute of Technology and Applied Research

E-mail: [1]vilasini@psgitech.ac.in, [2]vaisnav21032001@gmail.com, [3]manujnanthan01@gmail.com, [4]klkr@psgitech.ac.in

Abstract

The algorithms that are used for encrypting the text information for security purpose are referred as encryption algorithms. They are widely employed for the application on data transmission over networks. This is seen as a necessary part of data transmission since confidentiality and data privacy are important factors. Different encryption algorithms have different complexities, speeds of execution, and degrees of security. The Advanced Encryption Standard (AES) is one of the algorithms acknowledged by the U.S. government and other major organizations. The AES algorithm guarantees high security since there is no known public attack that can crack it. The Blowfish algorithm on the other hand is very fast but less secure. The encryption algorithms in the average zone of these two algorithms are highly useful for the transmission of general information among people. Such an algorithm also guarantees a good amount of speed as well as more security. Non-conventional algorithms such as transpositional cipher based on Rubik's Cube mechanism can be seen as a viable option to use in common applications. These algorithms can be further deepened with the help of more complex key generation parts that can be done to enhance the security of the data. This paper focuses on analytically evaluating if this algorithm can be treated as a viable replacement in an application compared to other algorithms. Moreover, this research covers some pros and cons noted while evaluating.

Keywords: AES, DES, Rubik Cube Algorithm, FCM

1. Introduction

Data security and privacy is an important criteria that needs to be handled with more care since secure information transmission is a very important factor. It is undesirable if in

DOI: https://doi.org/10.36548/jscp.2022.4.007
Received: 04.12.2022, received in revised form: 03.01.2023, accepted: 18.01.2023, published: 30.01.2023

any circumstances a hacker/third-party organization can tap out any information/data that is being sent in an encrypted manner. Users who are storing or sharing data with another person/third-party organization will try to ensure that the transferred data's confidentiality is maintained. At the same time, users want their data to be shared/stored as fast as possible to make sure the communication between the opposite end is smooth, fast, and secure as well. Every algorithm has its pros and cons and depending on the application's requirement one should take an architectural decision to prefer one over the other.

Unlike standard, well-known, and recognized algorithms, there isn't much data in transpositional cipher based on Rubik's cube's mechanism which is unconventional, that is tested with other algorithms in this paper. To test various encryption algorithms, a chat application is built on which the test bench for various encryption algorithms like Advanced Encryption Standard (AES), Data Encryption Standard (DES)/Triple DES, Blowfish, and Rubik's Cube algorithms are implemented on the encoding and decoding process where the performance of each algorithm is calculated based on preset metrics. The data which is either in the form of text, image, audio, or video, is sent as input from the sender side and is received exactly without any interference or any add-ons. The results are visually represented for easy understanding and are also theoretically calculated well enough to understand the advantages and disadvantages of each encryption algorithm.

2. Related Works

Paper [1] discussed the prevalence of end-to-end encrypted instant messaging services on mobile. It was said that end-to-end encryption makes it technically difficult for the secret services and government to combat terrorists, organized crime, and child pornographers. Governments have stated that they will only use the "backdoor" if a genuine threat to national security exists. Governments would like a "backdoor" in such apps to provide message access.

Article [2] discussed the use of algorithm decryption and encryption of images, text files, and audio in OpenSSL, as well as time analysis for data execution. 128-bit key length is used in AES 128 for decryption and encryption of block messages, AES-192 is equipped with 192-bit key length, and AES-256 is built with 256-bit key length. Each cipher encrypts and decrypts data in 128-bit blocks using cryptographic keys of 128, 192, or 256 bits. This demonstrated how the AES algorithm is used effectively in the open ssl program for the encryption and decryption of common text, files, and images. Improved security over unauthorized access was provided. It also specifies the amount of time required for the securing process. This work will be expanded to include data such as audio and video.

"Secure Image Encryption Algorithm based on Rubik's Cube Principle" was the foundation of the study [3]. The paper proposed a novel image encryption technique. The method for permuting image pixels was based on the Rubik's cube theory. The XOR operator muddles the link between the original and encrypted images by applying a key to odd rows and columns of a picture. The image's rows and columns were also treated with the same flipped key. The resistance of the algorithm to various types of attacks, including statistical and differential attacks, was tested experimentally using extensive numerical analysis (visual testing). Furthermore, performance evaluation experiments showed that the proposed picture encryption technique was extremely secure. It is trustworthy for real-time encryption and decryption.

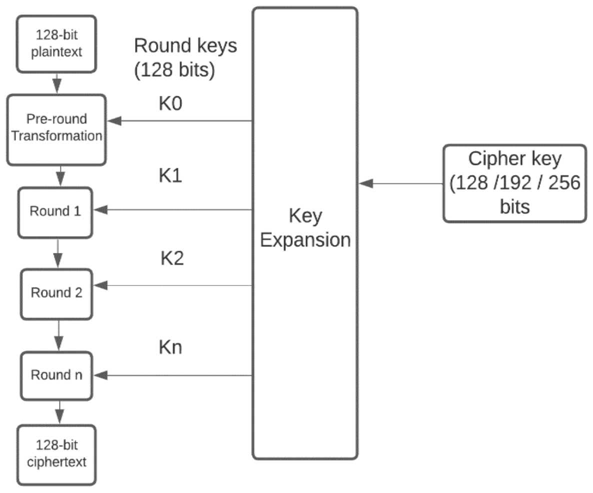


Figure 1. Process of AES

In [4], several encryption algorithms were compared for data security. For better data security and information security, a variety of encryption algorithms and techniques were compared. Comparisons of encryption algorithms based on performance, key size, software and hardware effectiveness, accessibility, implementation methods, and speed were analyzed. The study used Eclipse IDE to compare the reported encryption speed with a number of common algorithms provided by Oracle JDK, and it subsequently provided a summary of a number of other features of those techniques. AES (with 128 and 256-bit keys), DES, Triple

DES, IDEA, and Blowfish were the encryption methods taken into consideration (with a 256-bit key).

3. Proposed Methodology

The Rubik's Cube has 6 flat faces when represented on a 2-D flat surface. Each flat face has 3x3 cells contributing to 9 cells per face. The total number of cells will be 9*6=54 cells. In the encryption algorithm, a 3-D array of dimensions like the following is used.

int [6][3][3]

So here, the 1st index refers to the color ranging from White, Red, Green, Orange, Blue, and Yellow. The 2nd index refers to the row of the particular-colored face and the 3rd index refers to the column of the colored face.

There are 9 different rotational motions possible in the cube. Each rotation can be done in 3 ways, a single turn, a double turn, and a triple turn. The fourth turn will be the starting position itself. The representation of these rotations is by default in a clockwise direction. Anti-clockwise rotations are skipped because they can be derived through clockwise rotation itself. For example, one anticlockwise rotation can be made equivalent to a triple clockwise turn.

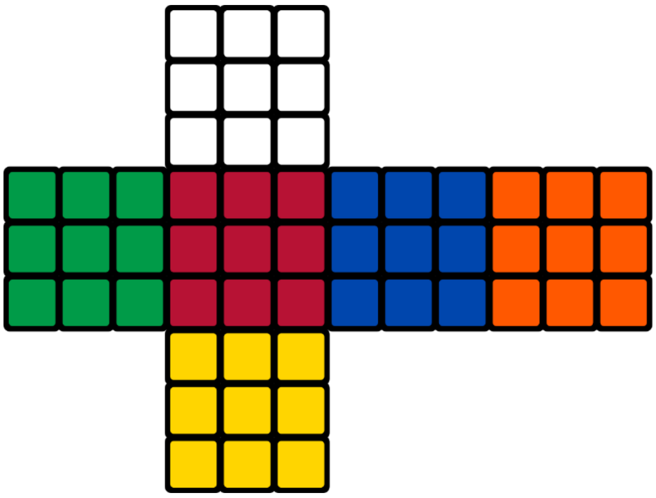
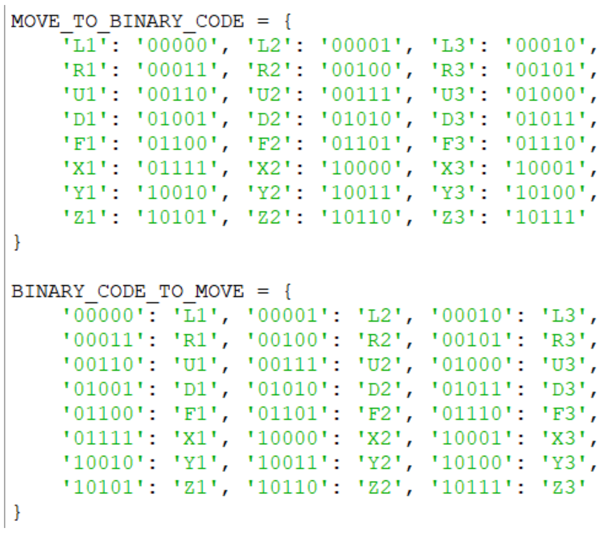


Figure 2. Representation of the 2D Rubik's Cube Structure

Since the total number of cells is limited to 54, the input data/text is split into block sizes of 54. Necessary padding and unpadded are done when the input size is less than/more than the block size. For each possible rotation, the shifted values are reassigned that vary according to the axis of rotation made. Each rotation is represented using a 5-bit binary number. The binary representation of all the rotations is concatenated and sent as a key to the receiving end in decimal form.

At the receiving end, the number is retrieved and converted back to the binary representation, and padding is done and split into multiples of 5. The rotation mappings are retrieved and their counter-rotations are performed to recover the original data sent from the user.

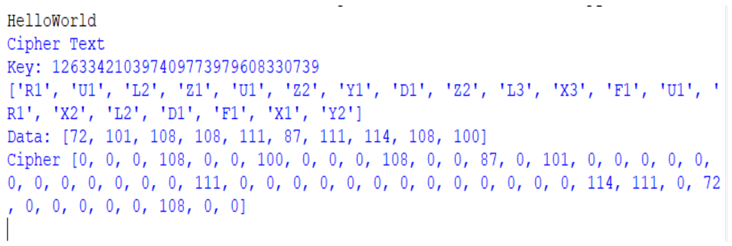


```
MOVE_TO_BINARY_CODE = {
    'L1': '00000', 'L2': '00001', 'L3': '00010',
    'R1': '00011', 'R2': '00100', 'R3': '00101',
    'U1': '00110', 'U2': '00111', 'U3': '01000',
    'D1': '01001', 'D2': '01010', 'D3': '01011',
    'F1': '01100', 'F2': '01101', 'F3': '01110',
    'X1': '01111', 'X2': '10000', 'X3': '10001',
    'Y1': '10010', 'Y2': '10011', 'Y3': '10100',
    'Z1': '10101', 'Z2': '10110', 'Z3': '10111'
}

BINARY_CODE_TO_MOVE = {
    '00000': 'L1', '00001': 'L2', '00010': 'L3',
    '00011': 'R1', '00100': 'R2', '00101': 'R3',
    '00110': 'U1', '00111': 'U2', '01000': 'U3',
    '01001': 'D1', '01010': 'D2', '01011': 'D3',
    '01100': 'F1', '01101': 'F2', '01110': 'F3',
    '01111': 'X1', '10000': 'X2', '10001': 'X3',
    '10010': 'Y1', '10011': 'Y2', '10100': 'Y3',
    '10101': 'Z1', '10110': 'Z2', '10111': 'Z3'
}
```

Figure 3. Mapping of Different Rotations

The above code represents the mapping of the rotations possible. These rotations are purely based on the clockwise representation only. The input data is supplied as a parameter along with other fixed parameters like maximum data length and other random initialization values. The key and the decrypted data are shown as output to the user.



```
HelloWorld
Cipher Text
Key: 126334210397409773979608330739
['R1', 'U1', 'L2', 'Z1', 'U1', 'Z2', 'Y1', 'D1', 'Z2', 'L3', 'X3', 'F1', 'U1', '
R1', 'X2', 'L2', 'D1', 'F1', 'X1', 'Y2']
Data: [72, 101, 108, 108, 111, 87, 111, 114, 108, 100]
Cipher [0, 0, 0, 108, 0, 0, 100, 0, 0, 0, 108, 0, 0, 87, 0, 101, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 111, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 114, 111, 0, 72
, 0, 0, 0, 0, 0, 108, 0, 0]
```

Figure 4. Sample Details of Input Text

4. Implementation Details

The real-time chat application was developed using Android Studio and its backend was developed using Node.js and MongoDB. The backend was hosted in AWS Cloud in an

EC2 instance. The chat was locally handled in the Android app using SQLite database. The main reason to develop a real-time application was to compare the impact of robustness and efficiency on user experience. However, to compare the two algorithms, there is a need for analytical data that a user experience can't provide. Hence Testbench code was developed that would put all the encryption algorithms to a test with the same data in the same environment. Various solutions for evaluation were explored. PSUTIL module in python was used but the results were unreliable at times, so the test bench was implemented in docker containers. A service was developed that allows one to submit a file via an exposed API. A query API was written to get the analytical data.

Docker containers are an excellent way to isolate processes on a single host. This is especially useful when running multiple CPU or memory-intensive processes, as it ensures that one process does not deplete the resources of the others. It is also possible to restrict how much CPU or memory a Docker container can use. This can be useful to prevent a single process from consuming too many resources on the host or to ensure that a container always has a minimum amount of resources available to it. Resources utilization can be monitored.

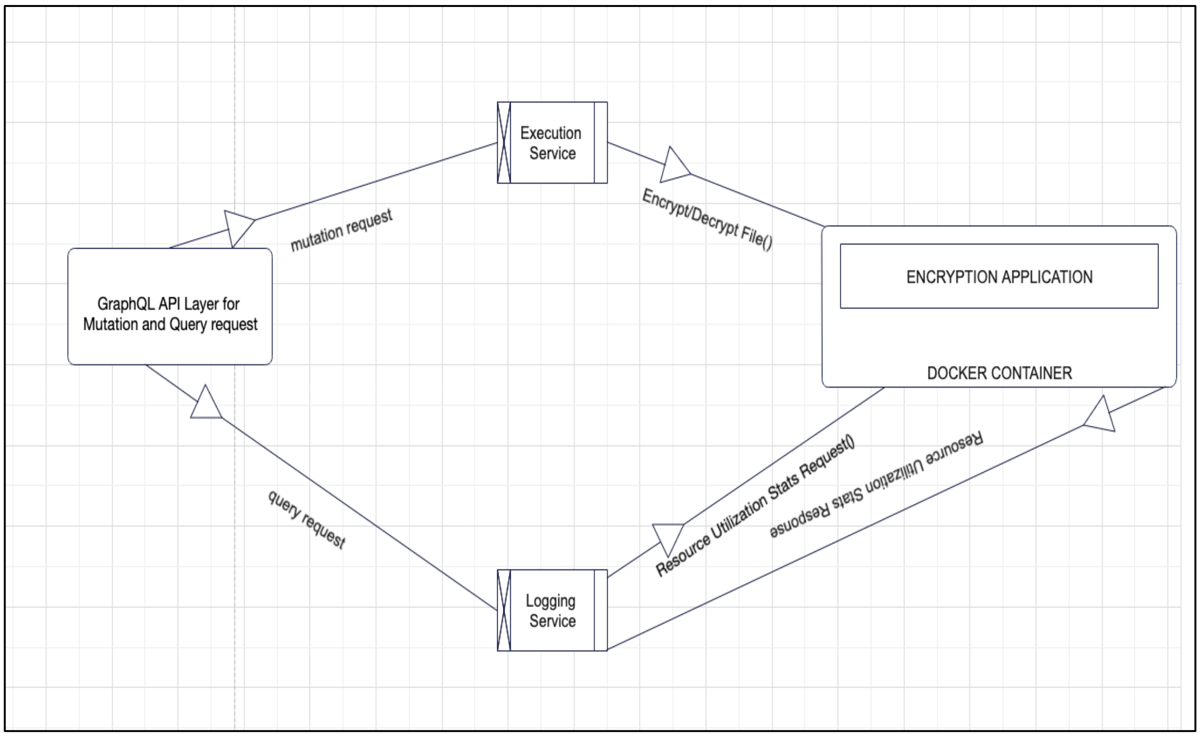


Figure 5. Architectural diagram representing how the test bench is implemented

A control group (cgroup) is a feature of Linux kernel that regualtes, isolates and accounts for the resource usage of a group of processes (Network, CPU, disc I/O, memory, etc.). So, basically, cgroups are used to control how much of the given key resource (CPU, memory, network, and disc I/O), a process or set of processes can access or use. Cgroups are an important component of containers because there are frequently multiple processes running in a container that must be controlled simultaneously. A logging service was written and used to monitor resource utilization. During the execution of any task, the resource utilization would be recorded, and this data is analyzed and is presented during the query to obtain analytical information of the execution. The GraphQL API layer runs in a service locally that processes requests submitted and handles the execution service and logging service.

5. Evaluation Metrics

5.1 Encryption Time

Encryption time is the amount of time required to transform plaintext into ciphertext. Key size, plaintext block size, and mode all affect how quickly data is encrypted. The encryption time in the experiment was measured in milliseconds. The system's performance is impacted by encryption time. Less time is consumed for encryption, resulting in a quick and responsive system. The number of rounds is proportional to encryption time. Block size is inversely proportional to encryption time when a whole file is considered, but over a single block, it is directly proportional.

Table 1. Encryption Time (in ms)

	1 MB	5 MB	10 MB	15 MB	20 MB
AES	4	20	36	56	76
DES	8	70	131	208	268
Blowfish	9	47	95	143	195
Rubik 3*3 (Key size = 50 b)	8	34	84	123	159
Rubik 3*3 (Key size = 130 b)	9	45	93	143	175
Rubik 5*5 (Key size = 130 b)	5	25	51	80	97

5.2 Decryption Time

The Decryption time is the amount of time required to convert ciphertext back to plaintext. To make the system responsive and quick, it is preferred that the decryption time be less similar to the encryption time. System performance is impacted by decryption time. The decryption time in the experiment is measured in milliseconds. The number of rounds is proportional to the decryption time. Block size is inversely proportional to decryption time when a whole file is considered, but over a single block, it is directly proportional.

Table 2. Decryption Time (in ms)

	1 MB	5 MB	10 MB	15 MB	20 MB
AES	2	16	30	47	62
DES	8	56	106	173	218
Blowfish	5	27	52	78	108
Rubik 3*3 (Key size = 50 b)	8	34	85	123	157
Rubik 3*3 (Key size = 130 b)	9	45	93	140	175
Rubik 5*5 (Key size = 130 b)	5	25	51	77	97

5.3 Memory Used

The amount of memory needed to implement various encryption algorithms varies. The algorithm's required number of operations, the size of the employed keys, the initialization vectors, and the type of operations all affect how much memory is needed. The amount of memory consumed affects the system's cost. It is preferred that the amount of memory used be as minimal as feasible. The number of transpositional operations increases the memory consumed, whereas mathematical operations don't consume additional space.

Table 3. Memory Consumption

Algorithm	Memory used (KB)
DES	18.2
AES	14.7
Blowfish	9.38
Rubik	40.3

5.4 Avalanche Effect

Diffusion is a cryptographic property that assesses the security of an algorithm. A small change in input results in a large difference in output. Another name for this is the avalanche effect. The avalanche impact is calculated with hamming distance. The hamming distance is used in information theory to quantify dissimilarity. The hamming distance is calculated as the sum of bit-by-bit xor when taking the ASCII value because it is simple to do programmatically. A high diffusion rate or an avalanche effect is preferred. The avalanche effect demonstrates the effectiveness of a cryptographic algorithm. Transpositional operations have little effect on the avalanche effect, whereas mathematical operations have more. When these two are combined, the avalanche effect is greatly increased.

5.5 Entropy

The unpredictability that an application gathers for use in cryptography necessitates the need for random data. A lack of entropy may have an adverse effect on performance and security. Entropy is critical to the strength of any cryptographic algorithm because it adds randomness to the cipher. An algorithm's entropy can be increased by defining a combination of mathematical, substitution, and transpositional cypher models.

6. Experimental Results

The graphs in Fig. 6 and Fig. 7 show how several encryption techniques, such as AES, DES, Blowfish, Rubik 3*3 with a key size of 50 bits, Rubik 3*3 with a key size of 130 bits, and Rubik 5*5 with a key size of 130 bits, take different amounts of time to encrypt data. The input file size or plain text provided to the encryption method is shown on the X-axis, and the time it took the encryption algorithm to construct the cipher for the input that was first passed is shown on the Y-axis.

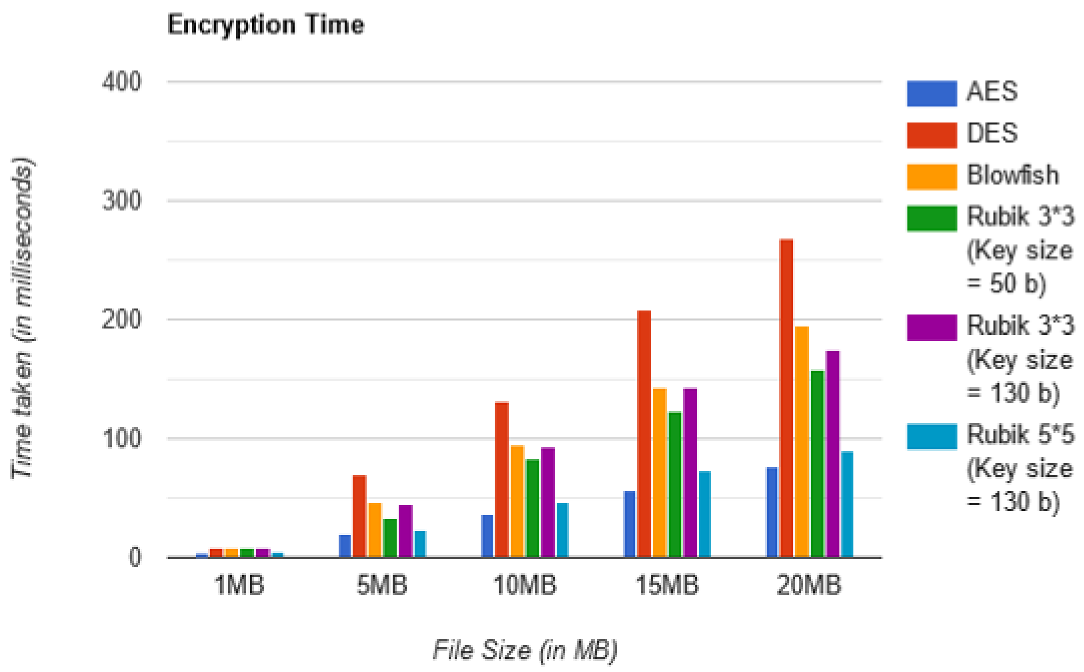


Figure 6. Plot representing encryption time vs file size

All these results were obtained by running these tests on a Docker container running an encryption service. The resources were limited to (1 CPU, 512Mb of memory). As can be seen, the temporal complexity of the encryption methods causes differences in how long they take to complete. It is clear from the above graphic representation that the AES method is quicker than the other algorithms described. In spite of this, there aren't many variances when the input data/size is minimal. It is observed that the discrepancies between the encryption techniques are considerably bigger when the input size is raised.

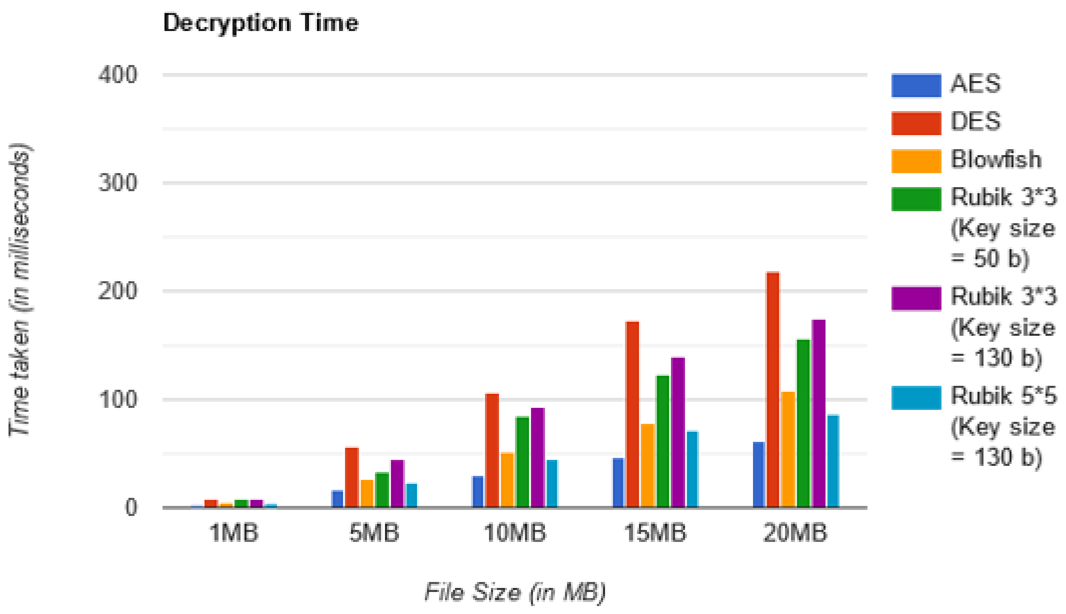


Figure 7. Plot representing decryption time vs file size

The Rubik 5*5 transpositional cipher with a 130-bit key size is the next fastest algorithm after the other ones discussed above. However, there isn't much of a difference between these two algorithms. The 50-bit key size Rubik 3*3 follows Rubik 5*5. Compared to the Rubik 3*3 with a 130-bit key size, this is a little faster. With a 130-bit key size, blowfish and Rubik 3*3 are pretty comparable. The second-slowest algorithm in terms of encryption time is blowfish. The DES algorithm is the slowest of all algorithms, according to the testing findings. Rubik 5*5 with a 130-bit key size is faster than any Rubik 3*3 mainly due to the increased block size. Though the computation performed in Rubik-based transpositional cipher is less complex than AES, AES is faster mainly because of two reasons:

- Better implementation of AES as the functions from python crypto-modules such as pycryptodome, are directly used.
- The block size of AES is 128 bits and the block size of Rubik 3*3 is 54 bits. So, in a way, Rubik 3*3 has to perform its computation 2.3 times to encrypt the same amount of data as AES in one cycle. This is the reason why Rubik 5*5 is faster than Rubik

3*3 and it matches the performance of AES, the main reason being its 150-bit block size.

Fig. 8. shows that AES provides the higher avalanche effect, whereas Rubik and RSA projects the reduced avalanche effect. This made an attention to look back over the AES for further analysis.

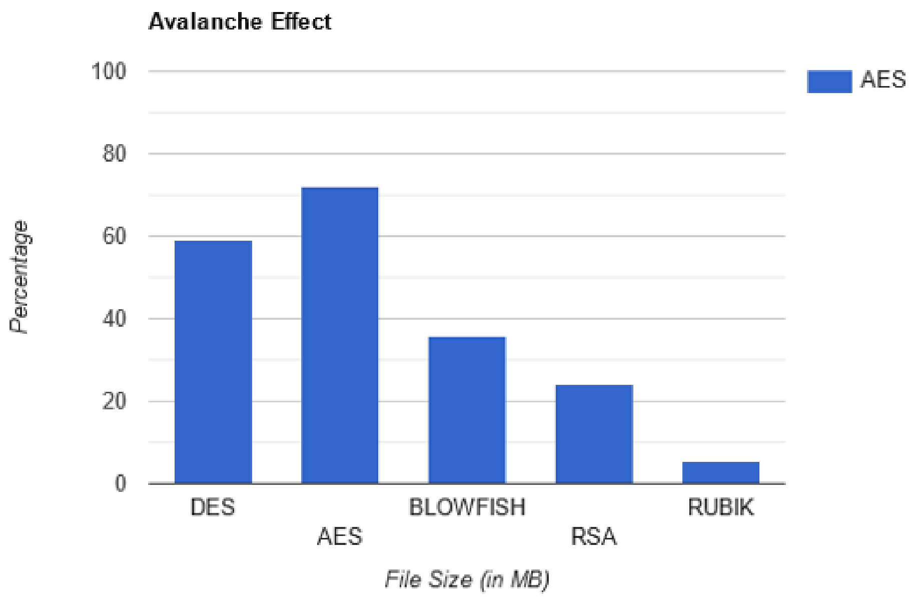


Figure 8. Plot representing avalanche effect in %

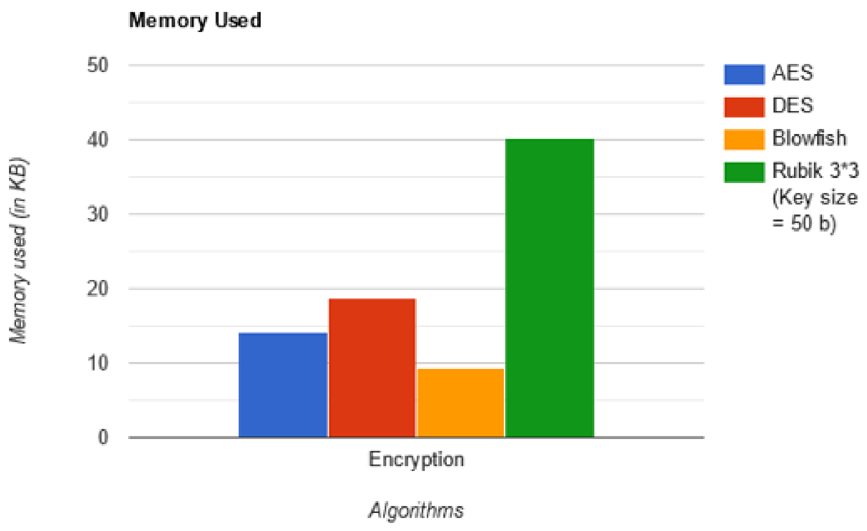


Figure 9. Memory consumed (in KB)

Generally, encryption algorithms with an avalanche effect of more than 50% are good encryption algorithms as it becomes too difficult to guess the encryption pattern. According to this, DES and AES only qualify as good encryption algorithms and AES is superior among these two. Rubik's Cube-based transpositional cipher has a low avalanche effect because it is a transpositional cipher and just certain values/bits are re-arranged. As computations are not performed in the work, the bit/values will not be affected other than the ones that are part of the plain data. However, generating random keys to encrypt frequently might help to overcome this issue as it will make writing a constructive algorithm difficult. This can be seen as a similar use case to that of an enigma machine.

All of these results were obtained by running these tests on a Docker container running an encryption service. The memory utilized has been computed using the help of the logging service which uses cgroups. The memory utilized by Rubik's Cube-based algorithm is high when compared with others based on a number of transpositions. For every transposition, a copy of the initial destination value is made, so that it would not get lost if it is required in a future transposition in the same block. Due to this, it consumes a lot of memory; however, the programming practice used is not the most memory efficient one. So, there is some room for improvement here.

7. Conclusion and Future Work

Based on the comparison results, a higher dimension Rubik's cube algorithm might serve as a viable replacement for AES based on the performance (encryption/decryption). However, from what has been implemented till now, it's not memory inefficient and therefore it can be memory intensive. Since it has a very low avalanche effect of 5.5%, it faces a great risk of guessing attacks. However, this risk can be eliminated by generating a new key at regular intervals. From the results attained, it is promising to say that the Rubik's Cube algorithm can be used as a replacement for other algorithms like Blowfish or regular DES. A Rubik Cube algorithm of lower dimension like 3*3, due to its lower block size can definitely act as a stream cipher. When all these algorithms were used in the crypto module of the chat application, any performance-related issues weren't noticed. However, from a practical use's perspective, this is subjected to change when data sharing like high-quality media content and documents is introduced.

References

[1] Robert E. Endeley, "End-to-End Encryption in Messaging Services and National Security—Case of WhatsApp Messenger", Journal of Information Security, Vol.09 No.01, 2018

[2] IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 22, Issue 6, Ser. I (Nov. – Dec. 2020), PP 39-44 www.iosrjournals.org

[3] Khaled Loukhaoukha, Jean-Yves Chouinard, Abdellah Berdai, "A Secure Image Encryption Algorithm Based on Rubik's Cube Principle", Journal of Electrical and Computer Engineering, vol. 2012, Article ID 173931, 13 pages, 2012. https://doi.org/10.1155/2012/173931

[4] Nazeh Abdul Wahid MD, Ali A, Esparham B, Marwan MD (2018) A Comparison of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish for Guessing Attacks Prevention. J Comp Sci Appl Inform Technol. 3(2): 1-7. DOI: 10.15226/2474-9257/3/2/00132

[5] E. Fernando, D. Agustin, M. Irsan, D. F. Murad, H. Rohayani and D. Sujana, "Performance Comparison of Symmetries Encryption Algorithm AES and DES With Raspberry Pi," 2019 International Conference on Sustainable Information Engineering and Technology (SIET), Lombok, Indonesia, 2019, pp. 353-357, doi: 10.1109/SIET48054.2019.8986122.

[6] Priyadarshini P, Prashant N, Narayan DG, Meena SM. A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish. Procedia Computer Science. 2016;78:617-624.

[7] Yogesh K, Rajiv M, Harsh S. Comparison of symmetric and asymmetric cryptography with existing vulnerabilities and countermeasures. International Journal of Computer Science and Management Studies. 2011;11(3):60-63.

[8] Elminaam DSA, Kader HMA, Hadhoud MM. Performance Evaluation of Symmetric Encryption Algorithms. International Journal of Computer Science and Network Security. 2008;8(12):280-286.

[9] Bono SC, Green M, Stubblefield A, Juels A, Rubin AD, Szydlo M. Security analysis of a cryptographically- enabled RFID device. In: SSYM’05: Proceedings of the 14thconference on USENIX Security Symposium. 2005.