A File Encryption Algorithm Based on Dynamic Block Out of order Matrix Mapping

1st Tan Dong
Department of Technical
Chongqing Radio and Television Digital Media Co., Ltd.
Chongqing, China
852041173@qq.com

2nd YanXia Wang
Chongqing Research Center for Information and AutomationTechnology
Chongqing Academy of Science and Technology
Chongqing, China
260789093@qq.com

3rd Liu Lei
The school of advanced manufacturing engineering
Chongqing University Of Posts And Telecommunications
Chongqing, China
1298803671@qq.com

***Abstract*—In order to solve the security of content transfer during file transfer and fast encryption of large files. This paper proposes a method to quickly encrypt large files and protect the contents of file information, also prevent files from being intercepted and decrypted before use. The algorithm of file encryption based on dynamic block out of order matrix mapping. First, dynamically block the files according to a certain ratio, extract the raw data and key information of each block, then store the out of order data into a two-dimensional table of matrix after calculation and processing. Second, use Base64 encoding on this matrix mapping table. Then the matrix mapping table is to be a unique decryption key of this file. After that, the data in the spare block of the file is filled with the MD5 data, the MD5 data is generated by UUID and block with the same size as the number of dynamic block. The new encrypted file whose length is equal to the original file. Because it is dynamic block and random extracted data, so it is fast and efficient. The experimental results and security analysis show that the algorithm has a good encryption structure and high security, which can effectively resist various attack behaviors. Most of the files that with different types can be used, and the speed to encrypt and decrypt large files is greatly improved compared with the traditional encryption and decryption methods, the speed and efficiency far exceed the traditional file encryption and decryption algorithms, such as AES, DES, 3DES and so on.**

Keywords*—*Dynamic block, Large file, Encryption, Out of order Matrix, Mapping

I. INTRODUCTION

At present, the encryption methods of common files mainly adopt symmetric encryption algorithms, such as AES , DES, 3DES and so on [1]. Symmetric encryption is fast, simple, with small calculation, and uses the same key for encryption and decryption. This method has obvious advantages in encrypting small files or text messages, but for the medium-sized or large files the process of encryption is slow and inefficient. In addition, both the encrypting side and the decrypting side of the file need to share and use the same key [2]. If there is leakage for any side, then the encrypted information is not safe any more, which is not safe for both side. And also there is a problem with the uniqueness of this key, which may be duplicated [3].

Therefore, in order to solve the above problems, this paper proposes a file encryption algorithm based on dynamic block out of order matrix mapping: dynamically block files according to a certain random ratio, extract 4 bits of raw data between each blocks. The MD5 value generated by UUID is divided into the same proportion of blocks for data replacement. The extracted raw data, indexes and weights of relevant data are calculated according to certain rules to form a block of data to be mapped into the matrix in disorder. And then conduct the Base64 coding to the disordered matrix data so that the matrix is equivalent to the unique decryption key of the file.

II. CONSTRUCT ENCRYPTION ALGORITHM

A. Dynamic Block with random position of File

The file to calculate the number of blocks by use file length(Byte) mod 1024 3 powers, Then multiply by 8. It can be calculated by the following formula:

$$B = \begin{cases} 8 & L < 1024^3 \\ \left(L/1024^3\right)\times 8 & L \geq 1024^3 \end{cases} \tag{1}$$

Where L is the length of the file (Byte) and B is the calculated number of blocks.

After determining the number of blocks, and then randomly determine the range of extracted data positions of each block. Specifying that the file header and the file tail are fixed extracted data blocks [4], the other B-2 blocks determine the position by generating random numbers. And the size of each block is 4 bytes, so the random number range is (4, L-4).

The formula for generating random numbers is the Linear Congruence Method [5]:

$$X_{n+1} = (aX_n + c)\ mod\ m, \quad n \geq 0. \tag{2}$$

Among them,

$$m, the\ modulus; \quad 0 < m.$$
$$a, the\ multiplier; \quad 0 \leq a < m.$$
$$c, the\ increment; \quad 0 \leq c < m.$$
$$X_0, the\ starting\ value; 0 \leq X_0 < m.$$

The structure resulting from the final block is roughly as follows:

0		1		2	……	B-1		B

Fig. 1. The Structure of Blocks

B. Generate replacement data blocks

Then generate a UUID and remove the - separator [6] in the UUID. MD5 is performed on the UUID, and the 32-bit string of the UUID is divided into blocks, also the 32-bit string of the obtained MD5 is also divided into blocks [7]. The UUID block data after is used as the weight value of each data in the matrix, and the data of the MD5 block is converted to the Byte to replace the raw data. Each block is 4 bytes in size.

C. Construct an out of order encryption matrix

First, the raw data of each block execute to add keyWeight, displacement and so on, to get a meaningful encryption data. Here is the following calculation method is used for encryption to generate encrypted data:

[block+index+100][blockIndex+index+100][data+index +100][start+keyWeight][keyWeight+10000000]

In this format:

131 127 218 163008429 15131510				
131	127	218	16300842 9	15131510
block + index +100	blockInde x +index +100	data+inde x +100	start+ keyWeight	keyWeight + 10000000

Fig. 2. Data calculated format

The block is the number of the block, index is the position index stored in the matrix, blockIndex is the index within the block (for example, the index position in the 4 Bytes raw data, from 0 to 3), the data is the raw data Byte value, and the start is the block start position in the file. The keyWeight is the weight value obtained by calculating the ASCII code of the character after the UUID grouping. In this way, an encrypted data string is obtained, and a total of B × 4 size data strings need to be calculated and placed into the matrix.

Next, a random number in the range of B × 4 (0, B × 4) is generated as an out of order storage postion in which the calculated B × 4 size data strings are stored in the matrix.

The format stored in the matrix is as follows:

TABLE I. ENCODED MATRIX STORAGE FORMAT

Index	**Row Data**			
	0	1	…	B-1
0	encryption data	encryption data	…	encryption data
1	encryption data	encryption data	…	encryption data
2	encryption data	encryption data	…	encryption data
3	encryption data	encryption data	…	encryption data

D. Generate a decrypted key

Finally, the two-dimensional matrix is converted into a string, and the 32-bit UUID is spliced at the end, then the final string encoded by Base64. So the string is the final decryption key.

The decryption operation algorithm step is also the inverse process of encryption [8]. The key step is to decrypt the data of each original file in the matrix, and then perform the replacement and recovery according to the block index position.

The flow chart of encryption is as follows:

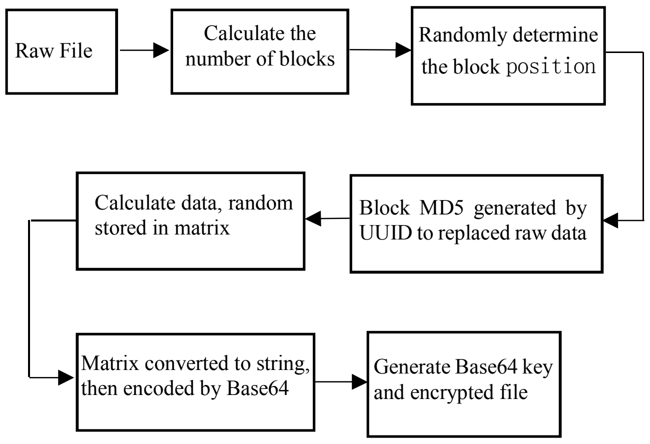
Raw File
Calculate the number of blocks
Randomly determine the block position
Block MD5 generated by UUID to replaced raw data
Calculate data, random stored in matrix
Matrix converted to string, then encoded by Base64
Generate Base64 key and encrypted file

Fig. 3. Encryption process

The flow chart of decryption is as follows:

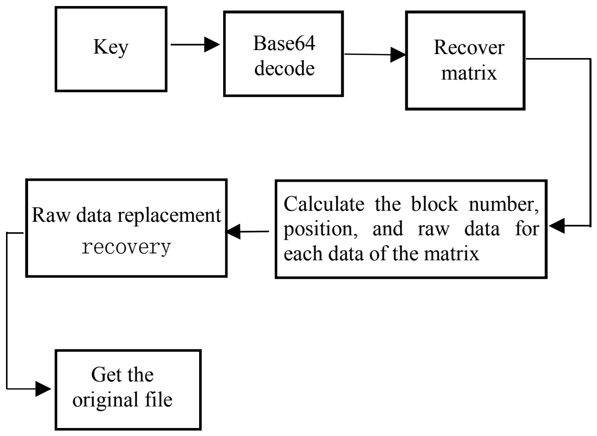
Key
Base64 decode
Recover matrix
Calculate the block number, position, and raw data for each data of the matrix
Raw data replacement recovery
Get the original file

Fig. 4. Decryption process

III. EXPERIMENTS AND RESULTS

Select the files of 187MB, 1187MB and 2385MB as the experimental data. Use the same size, and different file encryption algorithms to get the time consuming statistical comparison. Here, the typical mainstream symmetric encryption algorithm is selected: AES, DES, 3DES algorithm for statistical comparison with this paper algorithm.

The 187MB file is dynamically divided into 8 random blocks by the formula (1). The random positions obtained in the experiment are: 0, 34481781, 173960708, 35428008, 122204283, 94115247, 77991145, 1973674040. The generated matrix size is 8×4 size, the generated key length is 1152 Byte. The encryption takes 595 milliseconds, and the decryption takes 24 milliseconds.

The figure uses DBS to represent the algorithm of this paper. The time consuming comparison statistics of the 187MB file is as follows:

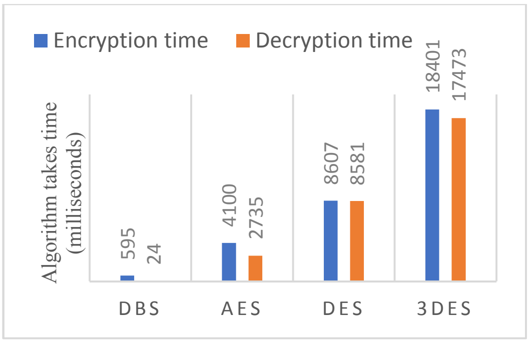

Fig. 5. 187MB file encryption and decryption time consuming statistical comparison

The 1187MB file is dynamically divided into 8 random blocks by the formula (1). The random positions obtained in the experiment are: 0, 1078708605, 323133824, 609375030, 763565877, 764770122, 917633321, 1252903205. The generated matrix size is 8×4 size, the generated key length is 1172 Byte. The encryption takes 360 milliseconds, and the decryption takes 28 milliseconds.

The figure same uses DBS to represent the algorithm of this paper. The time consuming comparison statistics of the 1187MB file is as follows:

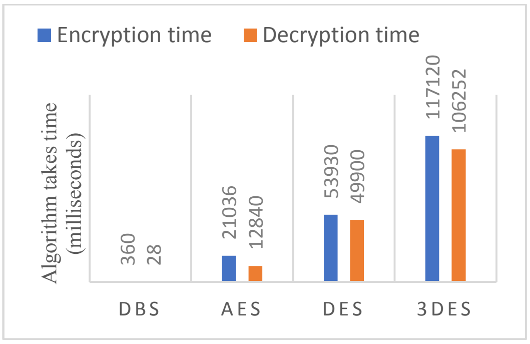

Fig. 6. 1187MB file encryption and decryption time consuming statistical comparison

The 2385MB file is dynamically divided into 2×8 random blocks by the formula (1). The random 2 × 8 positions obtained in the experiment are: 0, 713463986, 1418909879, 1951280390, 1965044600, 2096603066, 2499832095, 2443940873, 2289790361, 2237453225, 2171881527, 2157515025, 2122325312, 2151005447, 2145410690, 2505806394. The generated matrix size is 16×4 size, the generated key length is 2412 Byte. The encryption takes 425 milliseconds, and the decryption takes 27 milliseconds.

The figure same uses DBS to represent the algorithm of this paper. The time consuming comparison statistics of the 2385MB file is as follows:

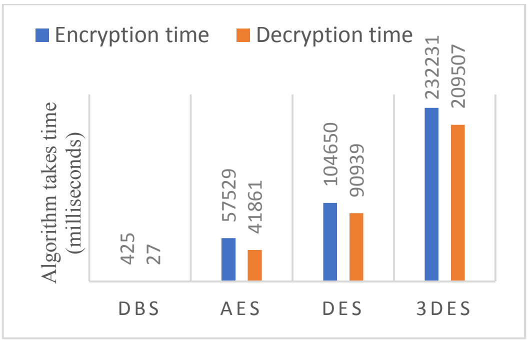

Fig. 7. 2385MB file encryption and decryption time consuming statistical comparison

Combining the three statistical charts, we can see that the encryption and decryption speed of the algorithm is much higher than the AES, DES, and 3DES algorithms, also the time spent on encryption and decryption is basically not affected by the file size.

IV. CONCLUSIONS

Through experiments and data analysis, the algorithm of dynamic block out of order matrix mapping in this paper is more effective and far less time-consuming in file encryption and decryption than AES, DES and 3DES algorithms. The larger the file is, the advantage of its speed and time spent on encrypting and decrypting will be more obvious. In addition, the time-consuming of encryption and decryption is basically unaffected by the size of the file. As it is dynamic block of the file, with the random extraction of raw data, the random selection of data block location, and out of order mapping after the confusion of data calculation into the matrix to form keys, security and efficiency can fully meet the function of cryptography.

REFERENCES

[1] Panda M, Nag A. Plain Text Encryption Using AES, DES and SALSA20 by Java Based Bouncy Castle API on Windows and Linux[C]//Advances in Computing and Communication Engineering (ICACCE), 2015 Second International Conference on. IEEE, 2015: 541-548.

[2] Singh G, Kinger S. Integrating AES, DES, and 3-DES encryption algorithms for enhanced data security[J]. International Journal of Scientific & Engineering Research, 2013, 4(7): 2058.

[3] Sivakumar R, Balakumar B, Pandeeswaran V A. A Study of Encryption Algorithms (DES, 3DES and AES) for Information Security[J]. 2018.

[4] Li Xiaodong, Zhou Cailan, Huang Linquan. A SECURE AND EFFICIENT IMAGE ENCRYPTION ALGORITHM BASED ON DNA CODING[J]. Computer Application and Softwre, 2018, 35(1): 318-324.

[5] Cicirello V A. Impact of Random Number Generation on Parallel Genetic Algorithms[C]//Proceedings of the Thirty-First International Florida Artificial Intelligence Research Society Conference. 2018: 2-7.

[6] Balkić Z, Šoštarić D, Horvat G. GeoHash and UUID identifier for multi-agent systems[C]//KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications. Springer, Berlin, Heidelberg, 2012: 290-298.

[7] Binti Suhaili S, Watanabe T. High-Throughput Message Digest (MD5) Design and Simulation-Based Power Estimation Using Unfolding Transformation[J]. Journal of Signal Processing, 2017, 21(6): 233-238.

[8] Chai X, Chen Y, Broyde L. A novel chaos-based image encryption algorithm using DNA sequence operations[J]. Optics and Lasers in engineering, 2017, 88: 197-213.