RESEARCH ARTICLE OPEN ACCESS

Application of RSAA for File Encryption

*Vishal Snedan Robertson [1], Krishnakant Vilas Redkar [2]
[1] (Student B.E. Computer Engineering, AITD, Goa, India.
[2] (Student MScIT, PRIDE, Periyar University, Salem, Tamil Nadu, India
*Corresponding Author: * Vishal Snedan Robertson*

ABSTRACT
Security of files for storage or transmission has become extremely important in this age. Files are used all around us. Prevention of certain files from unauthorised access is very important. Personal or critical files need to be safeguarded against intruders and hackers. We put forth a variable key length symmetric cryptosystem to encrypt text files based on the RSAA algorithm. It encrypts files by dividing its contents into blocks of 256 bits block at a time. High security is achieved even with a single character key. This algorithm is fast and can be used to encrypt large text files quickly with ease. Brute forcing this algorithm is infeasible due to Block Cellular Automata's immense rule space and since key size isn't fixed.

Keywords: File, Encryption, Decryption, Block Cellular Automata; Margolus neighbourhood; Rule Generator, Add Key, Hashing.

Date of Submission: 26 -06-2017 Date of acceptance: 20-07-2017

I. INTRODUCTION

The objective is to apply the RSAA algorithm to encrypt & decrypt text files. previous work, purpose, and the contribution of the paper. The 1st section explains the research done on the current algorithm. The 2nd section describes the designed algorithm. 3rd section shows some of the results obtained and the 4th section is for the conclusion.

II. LITERATURE STUDY

The RSAA algorithm is a symmetric key cryptosystem [1]. It uses a single secret key used by the sender to encrypt and by the receiver to decrypt. The secret key need to be shared between the communicating parties. The major operation in RSAA is based on Block Cellular Automata and its rules. It generates 16 Block Cellular Automata rules dynamically based on the input key using the RuleGen() function. The key entered by the user undergoes hashing in the HashKey() function. The encrypted output is in hexadecimal, whereas the input can be any ASCII characters. Input message is broken down into blocks of 256bits. The encryption takes place on this 256bit block. After 16 iterations wherein each iteration uses one of the 16 generated Block Cellular Automata rules, the encrypted message is shown as a sequence of 64 hexadecimal characters per input block.

III. DESIGNED CRYPTOSYSTEM

The designed algorithm uses functions defined in RSAA Symmetric Key Cryptosystem [1]. Functions such as RuleGen(), HashKey(), AddKey(), Encrypt Text() and DecryptText() are used as they are defined by the authors. Changes to the encryption and decryption function are made so as to encrypt and decrypt files.

2.1 Rule Generator

The RuleGen() function is applied on the output of the HashKey() function which takes in as input the key. The Rule Generation algorithm is as follows.

```
Algorithm RuleGen(Key)
numRules is the number of rules generated.
binKey stores the key in ASCII 8bit binary.
TempMat is a matrix of size numRules x 16
TempMat stores a 4bit hex representation of binKey.
RuleMat is a matrix of size numRules x 16.
RuleMat stores the final rules and is initialized to -1.
For every row of TempMat
index i is at start of the current row ie 0 and index j is at end of the current row ie 15
while i is less than j
Create a pair of the element at i and j which has not been paired before, if a pair can't be made then shift i to the right or j to the left or both.
Store element at j, at location i of current row of RuleMat
Store element at i, at location j of current row of RuleMat
```

For every location of current row of RuleMat whose value is -1, store value of location at location of current row of RuleMat
Return RuleMat

2.2 Hashing Function

A hash function is applied on the input key so as to create a key of constant length irregardless of the user input. Various hash functions can be used. SHA-512 is applied on the key as shown below.

```
Algorithm HashKey(key)
h1 = SHA-512 on key
h2 = SHA-512 on h1
Return h1 + h2
```

2.3 XOR function

The AddKey() function performs bitwise XOR on the bits in the block and corresponding bits of the key. The AdddKey() function is as shown below.

```
Algorithm AddKey(MessageMat,KeyMat)
For every bit of MessageMat
    MessageMat = KeyMat XOR MessageMat
```

2.4 Text Encryption

The function used to encrypt text is shown below.

```
Algorithm EncryptText(Message, Key)
Key = HashKey (Key)
RuleMat = RuleGen (Key)
RuleMat is a matrix of size 16 x 16
RuleMat stores the BCA rules generated by RuleGen
KeyMat is a matrix of size 64 x 16
KeyMat stores the bits of the Key after hashing
MessageMat is a matrix of size 16 x 16
MessageMat stores a current block of 32 characters
to be encrypted from Message
NumBlocks is the number of blocks that the
plaintext can be divided into
Pad Message with blank spaces to fill the last block.
For every block
Fill bits of current block into MessageMat
For every rule in RuleMat
Apply FBCA transformation on MessageMat
using current Rule
Apply AddKey () on MessageMat and KeyMat
using one of the 4 parts of the Key
Extract hexadecimal equivalent of the bits in
MessageMat and append it to cipherText
Return cipherText
```

2.5 Text Decryption

The function to decrypt text is shown below.

```
Algorithm DecryptText(Message, Key)
Key = hashKey (Key)
RuleMat = RuleGen (Key)
RuleMat is a matrix of size 16 x 16
RuleMat stores the BCA rules generated by RuleGen
KeyMat is a matrix of size 64 x 16
KeyMat stores the bits of the Key after hashing
CipherMat is a matrix of size 16 x 16
CipherMat stores a current block of 32 characters to be encrypted.
NumBlocks is the number of blocks that the plaintext can be divided into.
For every block
Fill bits of current block into CipherMat For every rule in RuleMat  Apply AddKey () on CipherMat and KeyMat using one of the 4 parts of the Key in reverse order  Apply RBCA transformation on CipherMat using current Rule Extract ASCII equivalent of the bits in CipherMat and append it to originalText
Return originalText
```

2.6 File Encryption

Files can be any type of text files from word files to program files. The file to be encrypted is read from the input path specified. Padding is done to make the length of the file a multiple of 32. After padding is applied the padded input is split into blocks of 256bit. Each block is encrypted using one of the 16 Block Cellular Automata rules. After all the blocks are encrypted, their output is combined to form the ciphertext, The FileEncryption() algorithm is given below.

```
Algorithm FileEncryption(Key, IP_Path, OP_Path )
Data = contents of file read using ipPath
Split Data into blocks of 256bit
For each 256bit block
Output = EncryptText(Data Block, HashKey(key)))
Append Output to file using opPath
```

2.7 File Decryption

The file to be decrypted is read from the input path specified. Each block is decrypted using one of the 16 Block Cellular Automata rules applied in reverse order. After all the blocks are decrypted, their output is combined to form the original message or plaintext, The FileDecryption() algorithm is given below.

```
Algorithm FileDcryption(Key, IP_Path, OP_Path )
Data = contents of file read using ipPath
Split Data into blocks of 256bit
For each 256bit block
Output = DecryptText(Data Block, HashKey(key)))
Append Output to file using opPath
```

IV. RESULTS

Table 3.1 depicts the results obtained for the designed algorithm. As can be seen, the encryption ^ decryption times are almost similar. Both the encryption and decryption times grow linearly with an increase in file size.

File Size in MB	Encryption Time in sec.	Decryption Time in sec.
1	3	3
2	7	6
5	15	14
10	30	30

Table 3.1

V. CONCLUSION

The designed algorithm is secure and robust, due to the use of the hashing function. It is fast and can be used to encrypt huge files in no time. A limitation for this algorithm is that it can be used only on text files and not image, audio or video files. Applications of this cryptosystem can be to securely store files on ones on system or to send encrypted files to other people. Servers can use this algorithm to store user's files after encrypting them, such that hackers get only encrypted files in the event of a leak.

REFERENCES

[1] Shreedatta Sawant, Vaishnavi Kamat, Vishal Snedan Robertson, Sneha Kamat, Anish Thali, and Anuj Shetgaonkar, RSAA Symmetric Key Cryptosystem, *IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 19, Issue 3, Ver. III,* May - June 2017, pg 53-57

[2] Said Bouchkaren and Saiida Lazaar, A Fast Cryptosystem Using Reversible Cellular Automata, *IJACSA International Journal of Advanced Computer Science and Applications, Vol. 5, No. 5,* 2014

Vishal Snedan Robertson . "Application of RSAA for File Encryption." International Journal of Engineering Research and Applications (IJERA) 7.7 (2017): 08-10.