IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 3, Ver. III (May - June 2017), PP 75-79
www.iosrjournals.org

RSA-A Symmetric Key Cryptosystem

Shreedatta Sawant[1], Vaishnavi Kamat[2], Vishal Snedan Robertson[3], Sneha Kamat[3], Anish Thali[3], Anuj Shetgaonkar[3]
[1] (Asst. Prof., Computer Engineering Department, AITD, Goa University, India)
[2] (Head, Asst. Prof. Computer Department, AITD, Goa University, India)
[3] (Students, Computer Engineering Department, AITD, Goa University, India)

Abstract: *Data security has become a growing concern in this digital age. Current cryptographic algorithms either compromise on computational speed or data security. We put forth a variable key length 1024-bit symmetric cryptosystem which encrypts and decrypts text data in the form of 256-bit blocks at a time. It is fast as well secure and it is completely dependent on the key.*

Keywords: *Cryptosystem; Encryption; Decryption; Ciphertext; Cellular Automata; Block Cellular Automata; Margolus neighborhood; Rule generator, Add key*

I. Introduction

Over the past few decades' wireless communication has become the most widely accepted form of communication. Wireless technology has made communication possible across the globe within seconds. But as to every advantage there is a disadvantage with the faster speed of communication comes the disadvantage of threat to security of the data which is transmitted over the wireless lines. Data which is transmitted over the wireless channel can be read and modified by a third party listening to the channel. This is a severe disadvantage as it may lead to loss and misuse of critical information sent over the communication line.

Encryption is one of the efficient ways used to secure data over the channel, in which messages are converted into human unreadable format such that only authorised people can get access to the original message. This message or information to be transmitted is known as plaintext and the message in human unreadable format is known as cipher text. For securing plaintext, various cryptographic systems are available which are divided into two categories Symmetric cryptographic systems and Asymmetric cryptographic systems. In a symmetric cryptographic system, a single key is used by the sender and the receiver to encrypt the plaintext and to decrypt the cipher text. Symmetric algorithm includes AES (Advanced Encryption Standard), Twofish, etc. In Asymmetric cryptographic system two separate keys are used for encryption and decryption of plaintext one of them is a shared public key and the other is a private key such as RSA.

II. Literature Survey

A cellular automaton is a collection of cells on a grid which changes its state with time depending on a set of rules & on the states of neighbouring cells. A block cellular automaton is a special kind of cellular automaton in which the grid of cells is split into non-overlapping blocks and the rules are applied to a whole block at a time. The neighbourhood of a cell is the nearby cells. One of the neighbourhoods used in block cellular automata is the Morgolus neighbourhood, in which the grid is divided into 2-cell blocks which are shifted by one cell along both axes on alternate time steps. Block Cellular Automata Rules are defined as a 1D array containing a permutation of the numbers 0 to 15.

In 2014, Said Bouchkaren and Saiida Lazaar [1] explain an algorithm to use Block Cellular Automata in cryptography. The algorithm uses a single rule.

Each iteration of transformation using the rule has 2 phases.

Phase 1: Create sub-blocks of size 2x2 starting from cell at row 0 & column 0.

Decimal equivalent of each block is obtained and is used as an index in the Rule to obtain the value to be mapped back to the block.

Phase 2: Create sub-blocks of size 2x2 starting from cell at row 1 & column 1.

Decimal equivalent of each block is obtained and is used as an index in the Rule to obtain the value to be mapped back to the block

Forward Block Cellular Automata (FBCA) performs in each iteration Phase 1 followed by Phase 2.

Reverse Block Cellular Automata (RBCA) performs in each iteration Phase 2 followed by Phase 1.

In 2004, Marcin Seredynski and Pascal Bouvry under the section Reversible Cellular Automata have stated that the number of reversible rules that can be used in cryptography should be a very large number and that they should behave in a complex manner. Elementary 1D CA has only 256 rules in total of which only 6 rules can be used for cryptography, as they are reversible. But BCA has 16! rules i.e. 2.092279E+13 rules of which there are numerous reversible rules that can be used to encrypt data. In "Theory and Appplications of Callular Automata in Cryptography" [3] The authors have proposed an algorithm wherein between the iterations an XOR is applied on the message and the key. This simple function introduces alot of randomness without increasing the overall time needed by the algorithm.

III. Proposed Cryptosystem

The proposed cryptosystem is based on block cellular automata and uses the Margolus neighbourhood. The approach is to have the algorithm use BCA rules that are generated dynamically and not be dependent on only the rules predefined in the algorithm. BCA rule space is 16! = 20922789888000 rules. The key can be anything and of any length. A single plaintext block is a sequence of 32 8bit ASCII characters for each input block. The message space is $16^{64} = 3.222\text{x}10^{74}$ different 1 block long messages can be entered by the user. The ciphertext is a sequence of 64 hexadecimal characters for each block that is being encrypted. Hence the cipher space is $16^{64} = 1.1579\text{x}10^{77}$ different ciphertext that can be generated.

The rules are created from the output of the hashKey() function. The secret key is a sequence of 256 hexadecimal characters (1024bits) and is used for encryption and decryption. 16 rules are generated from the key since every 8 characters in the key will generate a rule of block cellular automata. The proposed Rule generation algorithm is as follows.

1.Algorithm RuleGen(Key)

```
numRules is the number of rules generated (every 8 characters of the Key creates a rule) 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
```

An example of the Rule Generation algorithm is given below.
Key: function
Binary Equivaent: 01100110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
Split Binary Equivaent: 0110 0110 0111 0101 0110 1110 0110 0011 0111 0100 0110 1001 0110 1111 0110 1110
Hexadecimal equivalent: **6** 6 **7** **5** 6 **14** 6 **3** 7 **4** 6 **9** 6 **15** 6 **14**

Initially	: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
Pair 6,14	: -1 -1 -1 -1 -1 -1 **14** -1 -1 -1 -1 -1 -1 -1 **6** -1
Pair 7,15	: -1 -1 -1 -1 -1 -1 14 **15** -1 -1 -1 -1 -1 -1 6 **7**
Pair 5, 9	: -1 -1 -1 -1 -1 **9** 14 15 -1 **5** -1 -1 -1 -1 6 7
Pair 3, 4	: -1 -1 -1 **4** **3** 9 14 15 -1 5 -1 -1 -1 -1 6 7
Map rest	: **0** **1** **2** 4 3 9 14 15 **8** 5 **10** **11** **12** **13** 6 7

Key function (Rule): 0,1,2,4,3,9,14,15,8,5,10,11,12,13,6,7

Existing cryptographic algorithms which are based on cellular automata used predefined rules such as Rule 30 in 1D cellular automata algorithms and Rule {3,2,5,8,9,7,0,4,1,13,6,14,10,15,11,12} in block cellular automata algorithms. The proposed RuleGen creates 16 rules from the key entered from 16! possible rules.

The algorithm works in case of weak keys as it applies a hashing function on the key entered by the user. This makes the key given to the ruleGen as a fixed 1024bit key compared to a small, weak key entered by the user. A key of even a single character after being hashed becomes a 1024 bits long key. The hashing algorithm applied here is SHA-512 and is applied on the input key in the following manner

2. Algorithm HashKey (key)

```
h1 = SHA-512 on key
h2 = SHA-512 on h1
Return h1 + h2
```

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

3. Algorithm AddKey (MessageMat, KeyMat)

```
For every bit of MessageMat
MessageMat = KeyMat XOR MessageMat
```

The Encryption () function is provided an input of a plaintext and key in ASCII form and generates the ciphertext in hexadecimal form. It begins with the HashKey () and RuleGen () functions being applied on the Key. NumBlocks is size of Message divided by 32. Each block undergoes FBCA trnsformation followed by AddKey (). This continues for each of the 16 rules present in RuleMat to be applied in each iteration of the curent block. Once the block is encrypted its correcponding cipher value is appended to cipherText and the next block is taken up for encryption. After all blocks have undergone encryption the cipher text is stored in cipherText. The Encryption () function is as follows.

4. Algorithm Encryption (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
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
```

The Decryption () function takes the ciphertext in hexadecimal and key in ASCII form as input, and generates a plaintext in ASCII as output. It begins with the HashKey () and RuleGen () functions being applied on the Key. NumBlocks is size of Message divided by 64. Each block undergoes AddKey () followed by RBCA. This continues for each of the 16 rules present in RuleMat to be applied in each iteration of the curent blocks nin the reverse order. Once the block is decrypted its correcponding decrypted value is appended to originalText and the next block is taken up for decryption. After all blocks have undergone decryption the decrypted text is stored in originalText. The Decryption () function is as follows.

5. Algorithm Decryption (Ciphertext, 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
```

The functions used to encrypt and decrypt messages can also be used to encrypt and decrypt text files using the following algorithms.

6. Algorithm FileEncryption (InPath, OutPath, Key)

```
FileData is content read from file at InPath
FileData = Encryption (fileData,Key)
Store content of fileData in a file at OutPath
```

7. Algorithm FileDecryption (InPath, OutPath, Key)

```
fileData is content read from file at InPath
fileData = Decryption(fileData,Key)
Store content of fileData in a file at OutPath
```

RSAA can be applied on images as well. Images encrypted using RSAA all look similar hence a variety of different images after encryption looks similar. The image to be encrypted is read from the input path specified. Padding is done to make the dimensions of the image a multiple of 50. Hence 50x50 blocks of pixels of the image will be encrypted and decrypted at a time. After padding is applied the 24bit ARGB values of the image are read into arrays to be used for manipulation. The R, G and B components of a block undergo FBCA followed by an AddKey() step. This happens for all 16 rules generated. AddKey() uses a part of the key in each iteration. Similarly, every block is encrypted and once all are done, the cipher image is written to the output path specified. The ImageEncryption() algorithm is given below.

8. Algorithm ImageEncryption(InPath, OutPath, Key)

```
Key = HashKey(Key)
RuleMat = RuleGen(Key)
RuleMat is a matrix of size 16 x 16
Pad the image at InPath using a specific color so that the image can be divided into blocks of 50 x 50 pixels
Read RGB values of the image at InPath into Rmat, Gmat and Bmat
For each 50 x 50 block of image
        For every rule in RuleMat
                Apply FBCA transformation on Rmat, Gmat and BMat using current Rule
                Apply AddKey() on Rmat, Gmat and BMat and KeyMat using one of the 4 parts of the Key
Write RGB values from Rmat, Gmat and Bmat into an image at OutPath
```

The image to be decrypted is read from the input path specified. The 24bit ARGB values of the image are read into arrays to be used for manipulation. The R, G and B components of a block undergo AddKey() followed by a RBCA step. This happens for all 16 rules generated in reverse order of encryption. AddKey() uses a part of the key in each iteration also in reverse order of encryption. Similarly, every block is decrypted and once all are done, the decrypted image is written to the output path specified. The ImageDecryption() algorithm is given below.

9. Algorithm ImageDecryption(InPath, OutPath, Key)

```
Key = HashKey(Key)
RuleMat = RuleGen(Key)
RuleMat is a matrix of size 16 x 16
Read RGB values of the image at InPath into Rmat, Gmat and Bmat
For each 50 x 50 block of image
        For every rule in RuleMat
                Apply AddKey() on Rmat, Gmat and BMat and KeyMat using one of the 4 parts of the Key in
                reverse order
                Apply RBCA transformation on Rmat, Gmat and BMat using current Rule Remove padded
                pixels' values from Rmat, Gmat and Bmat
Write RGB values from Rmat, Gmat and Bmat into an image at OutPath
```

IV. Experimental Results

This section we present the results for the proposed cryptosystem based on trial runs. The below analysis is done only on text encryption and not on file and image encryption. The first test checks for the Strict Avalanche Criterion (SAC) which states that whenever a single input bit is inverted, each of the output bits' changes with a 50% probability or more probability. The key used for every test case is Two One Four Two.

Table.1. Avalanche Effect

Key	Plaintext without change	Stating T changed to U	Middle F changed to G	Ending o changed to p
Two One Four Two	11111111	100.00%	96.80%	90.22%
	12344321	93.75%	95.31%	89.06%
	aaaaaaaa	96.87%	93.75%	87.50%
	abcddcba	100.00%	93.75%	95.31%

As can be seen from the results, the amount of change in ciphertext based on a single character change in the key is always more than 85%. To learn the dependency of time on the message length we performed the next test. Here we encrypt messages of different block sizes using the same key.

Table.2. Time taken for Encryption and Decryption

Number of blocks	Encryption Time in sec.	Decryption Time in sec.
1	9	9
2	11	11
3	15	15
4	21	21

As seen from the above results, the time taken to encrypt is linear with respect to increase in the size of message.

V.Conclusion

The developed algorithm is a variable key size and 256bit block cryptosystem. Its security depends solely on the size of the key and the hashing algorithm used. Encryption and decryption speeds increase linearly as message length increases. The rules used to encrypt the message are dependent completely on the key & will differ from based on different keys. It also exhibits the avalanche effect, wherein a single bit change in input leads to more than 85% change in output.

References

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

[2] S. Nandi, B. K. Kar, and P. Pal Chaudhuri, "Theory and Applications of Cellular Automata in Cryptography" IEEE TRANSACTIONS ON COMPUTERS, VOL. 43, NO. 12, DECEMBER 1994.

[3] Marcin Seredynski and Pascal Bouvry, "Block Encryption Using Reversible Cellular Automata"

[4] Valloriw J. Peridier, "Basic Schemes for Reversible Two-Dimensional Cellular Automata", Complex Systems, 18, 2008, pg 47 48