ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 12, December 2016

# Implementation of REA Algorithm

[1] Akshay Kshirsagar [2] Anish Maniyar [3] Satish Phale [4] Siddhant Padve [5] Prof. Shubhada Mone
[1][2][3][4][5] Computer Engineering
Marathwada Mitra Mandal's College of Engineering
Pune, India.

***Abstract:*** **In today's world, heavy security is required to transmit confidential information over the network. Security is also demanding in wide range of applications. Cryptographic algorithms play a vital role in providing the data security against malicious attacks. Encryption of data is an important topic for research, as secure and efficient algorithms are needed that allow optimized encryption and decryption of data. In this paper we propose an implementation of an algorithm, called Reverse Encryption Algorithm (REA). Our encryption algorithm (REA) is simple and fast enough for most applications. REA encryption algorithm provides maximum security and limits the added time cost for encryption and decryption. This paper will focus on implementing sensitive data sets over a web application as per the user's choices.**

***Index words*** **— Security, encryption, decryption**

## I. INTRODUCTION

In the modern world, data has become extremely important. There are mainly two categories of data, that is, sensitive data and non-sensitive data. Data that can be shared with other people easily without any fear of it being misused, can be called as public data. Any form of multimedia on the Internet belongs to this category.
On the other hand, we can define sensitive data as the information that should not be made public. For example, military communications and projects are extremely confidential, and should be secure from the eyes of the general populace. Up until now, there have been many algorithms that attempt to secure data, such as the DES, AES, RSA and SHA to name a few. However, in this paper, we make use of a newer technique known as the REA Algorithm.

The basic aim of the REA algorithm is to achieve security in encrypting databases without degrading the performance of the application. It is fast enough to work on the web, and has little overhead in order to encrypt the data. While it is not the fastest or the most secure encryption algorithm, its unique technique allows it to strike a perfect balance between security as well as efficiency.

The practical approach until today has been to encrypt the entire database and decrypt it when the user sends a query. However, this increases overhead and minimizes efficiency of accessing the data. Therefore, in this paper, we discuss a model of encrypting only the data that the user wishes to keep private, while the rest of the data will be viewable to anyone who accesses it. Section 2 of this paper discusses about the use of databases to store sensitive and non-sensitive data. Section 3 will focus on the original REA algorithms used to achieve encryption and decryption. Our modifications to the original algorithms to increase security are shown in section 4. Section 5 will illustrate the use of the above algorithm. Our proposed system to use the above algorithms in a meaningful manner is showcased in section 6. Finally, section 7 presents the conclusion and future work that can be undertaken to improve the project even further.

## II. SELECTION OF DATABASES

Selection of an appropriate database plays as important a role in data security as selecting an appropriate algorithm does. The main aim of a database is to provide a method to store and retrieve data from a storage that is convenient as well as fast and efficient.

When databases are considered, the most popular choices are relational databases, as they provide a rigid support and schema for storing and accessing data. However, in the 21st century, the boom of digital media has brought about a change in this above trend.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 12, December 2016

Currently, unstructured databases are the trend, which are capable of storing data of any sort without the need for a complex structure.

Unstructured databases like Hadoop, Cassandra and MongoDB provide high performance and high availability of data with low overhead. In our proposed system, we have used MongoDB as our database, due to its open source nature, ease of use, and willingness to store data of any format in the form of documents.

MongoDB provides support for connectivity with Java, which can be easily used to create a front end for the database. Therefore, our proposed application will make use of the Java Struts Framework as a front-end, using MongoDB as a back-end.

## III. ORIGINAL REA ALGORITHM

The Reverse Encryption Algorithm, or REA, is being used due to its simplicity and efficiency. In this section, we will provide the algorithms used to encrypt as well as decrypt the data. The REA algorithm proposes using a key entered by the user and appending it to the input data. After the input and key are appended, the algorithm encrypts them together to generate a stream of cipher text. The algorithm is given below:

***REA Encryption Algorithm***
Step 1: Input the text and key.
Step 2: Add the key to the text.
Step 3: Convert the previous text to ASCII code.
Step 4: Convert the previous ASCII code to binary data.
Step 5: Reverse the previous binary data.
Step 6: Gather each 8 bits from the previous binary data and obtain the ASCII code from it.
Step 7: Divide the previous ASCII code by 4.
Step 8: Obtain the ASCII code of the previous result divide and put it as one character.
Step 9: Obtain the remainder of the previous divide and put it as a second character.
Step 10: Return encrypted text.

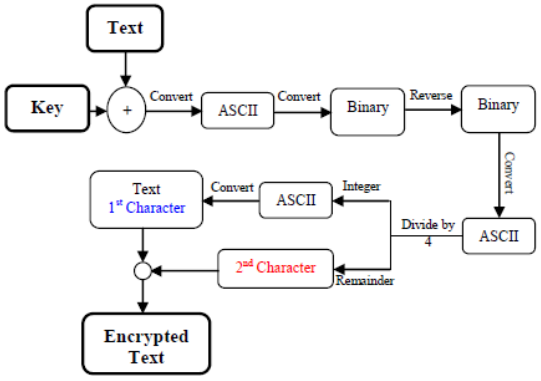



***Fig. 1: Control Flow of REA Encryption Algorithm***

Using the above algorithm, a simple input string can be converted into an undecipherable stream of numerals. Unless the key is known, no one can decipher this stream of numerals to find out the original data.

***REA Decryption Algorithm***
Step 1: Input the encrypted text and the key.
Step 2: Loop on the encrypted text to obtain ASCII code of characters and add the next character.
Step 3: Multiply ASCII code of the first character by 4.
Step 4: Add the next digit (remainder) to the result of multiplying operation.
Step 5: Convert the previous ASCII code to binary data.
Step 6: Reverse the previous binary data.
Step 7: Gather each 8 bits from the previous binary data and obtain the ASCII code from it.
Step 8: Convert the previous ASCII code to text.
Step 9: Remove the key from the text.
Step 10: Return decrypted data.

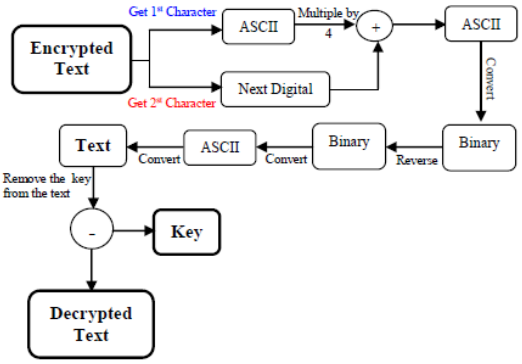



***Fig. 2: Control Flow of REA Decryption Algorithm***

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 12, December 2016

There is a huge security risk with the original REA Algorithm, which is explained in the next section.

## IV. PROPOSED MODIFIED ALGORITHM

The original REA algorithm falls short in securing data very well. In the original algorithm, the encryption key is simply appended to the input data, which can be easily figured out by an attacker. If an attacker were to remove the appended key from the input data, they would easily be able to view the private data.

Therefore, in order to secure the input string of data, we propose the following algorithm which customizes the encryption process by adding the key value to every input character. It is given below:

***Modified Encryption Algorithm***

1 – Generate a random key.
2 – Generate the ASCII value for each value of key and add the values to generate a single value.
3 – Convert the input text into their corresponding ASCII values.
4 – Convert each ASCII value to binary.
5 – Reverse the 8-bit binary value.
6 – Convert the reversed binary value to decimal.
7 – Add the ASCII value of the key to the previously generated decimal value.
8 – Divide the resulting value by 4.
9 – Convert the quotient of the division to its corresponding ASCII character.
10 – Append the remainder of the division to the ASCII character.
11 – The result is the cipher text of the input character.

Using a randomizer function, we generate a key that accepts both, alphanumeric characters as well as special symbols. The ASCII value of each value of this key is generated, and the values are summed to generate a final value, which will be used to encrypt the input data specified by the user.

After the binary reversal and conversion to decimal is done, instead of simply dividing the ASCII value by 4, we first add the final value of the key to each of the ASCII values of the input data. This gives us values which are completely different from the original data, and will not be decipherable without possession of the private key.

The remaining steps are similar to the original algorithm, and we get the final output as the cipher text which is completely unrecognizable from the input data. After the algorithm runs completely, the key used to encrypt the data is provided to the user, so that they can use the key in order to view their encrypted data whenever they so require.

Whenever an unauthorized user tries to view another user's data, they will only be able to access the data that is stored publicly, i.e. the data which the user did not specify to be encrypted. The encrypted data will be available to the unauthorized user as a stream of cipher text, which is unreadable.

If an authorized user wishes to view their private data, they are required to submit the encryption key provided to them at the time of encryption of data. If the key is correctly specified, the private data will be correctly deciphered, and displayed accordingly.

***Modified Decryption Algorithm***

1 – Each pair of characters in cipher text represents 1 input character.
2 – Convert the 1st character to ASCII code.
3 – Multiply it by 4 and add 2nd character to the result.
4 – Convert the key (accepted by user) into ASCII values and calculate their sum.
5 – Subtract the value from the previous result.
6 – Convert the result into 8-bit binary values.
7 – Reverse the binary values.
8 – Convert binary to corresponding values of ASCII.
9 – Convert the ASCII values to their corresponding characters.
10 – The result is the deciphered value of the cipher text.

The simplicity of the REA Algorithm is in that if a single character of the key is wrong, the entire string that is generated will be nonsensical. It will not provide the correctly decrypted data until the entire key is provided correctly.

In the next section, we take a look at an illustration to understand the process behind the conversion of the data from normal text to a stream of cipher text, as well as the decryption process to convert the data back into a human readable format.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 12, December 2016

## V. ILLUSTRATION

The process of encryption and decryption of data is quite lengthy and difficult to understand. Therefore, a simple example is provided below in order to understand the algorithm easily.

***Illustration - Encryption***

Assume a randomly generated key as 123 and assume a user input as "ABC".
Generate the ASCII value for each value of key and add them:

1-->49, 2-->50, 3-->51
Total = 49 + 50 + 51 = 150

Convert the input string to its corresponding ASCII code:

A-->65, B-->66, C-->67

Convert the previously generated ASCII code to 8-bit binary:

65-->01000001, 66-->01000010, 67-->01000011

Reverse the binary value:

01000001-->10000010, 01000010-->01000010, 01000011-->11000010

Convert the reversed binary value to decimal:

10000010-->130, 01000010-->66, 11000010-->194

Add the ASCII value of the key each of the generated decimal values:

130 + 150 = 280, 66 + 150 = 216, 194 + 150 = 344

Divide the resulting values by 4:

280/4 = 70, 216/4 = 54, 344/4 = 86

Convert the quotient of the division to corresponding ASCII character:

70-->F, 54-->6, 86-->V

Append the remainder to the generated ASCII character:

F-->F0, 6-->60, V-->V0

The result is the cipher text of the input value:

F060V0

As shown above, a simple text "ABC" can be converted into a stream of undecipherable text "F060V0". Now, let us consider the same example for decryption of data.

***Illustration – Decryption***

Each pair of corresponding symbols in cipher text represent one character:

F0-->First character, 60-->Second character, V0-->Third character

Convert the first symbol to ASCII code:

F-->70, 6-->54, V-->86

Multiply it by 4 and add the 2nd symbol to the result:

70*4 = 280 + 0(remainder) = 280
54*4 = 216 + 0(remainder) = 216
86*4 = 344 + 0(remainder) = 344

Convert the key accepted by user into ASCII value and calculate the sum:

1-->49, 2-->50, 3-->51
Total = 49 + 50 + 51 = 150

Subtract the value from the previous result:

280 – 150 = 130, 216 – 150 = 66, 344 – 150 = 194

Convert the result into 8 bit binary value:

130-->10000010, 66-->01000010, 194-->11000010

Reverse the 8 bit binary value:

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 12, December 2016

10000010-->01000001, 01000010-->01000010, 11000010-->01000011

Convert binary to corresponding value of ASCII:

01000001-->65, 01000010-->66, 01000011-->67

Convert the ASCII value to its corresponding character:

65-->A, 66-->B, 67-->C

The result is the deciphered value of the text:

ABC

Using the encryption key, we can successfully decrypt the cipher text to obtain our input data in a readable format. This encryption and decryption process works on any sort of input string since it converts the string into ASCII values which are standardized.

## VI. PROPOSED SYSTEM

For our proposed system, we are creating a web application which uses Java Struts Framework to integrate with the intermediary layer as well as the back-end. The forms will be designed using HTML and CSS, along with JSP for the client-server data handling.

In the first form, the user will specify the number of fields they require. Next, they will be redirected to a new form which will create the number of fields they have specified. The user will input the data they require into the fields on the form.

Next to each of these fields, there will be a radio button present, which will allow the user to specify whether the particular field is to be encrypted or not. By providing this option, we will greatly increase the granularity of the system, providing encryption only to the fields that the user themselves wishes to secure.

After submitting the data, the user will be prompted whether they wish to save their data or clear any field. After the user submits this form, the encryption algorithm will take over and encrypt the data. Following this, the data will be stored into the back-end.

Another form will allow the user to view, edit or delete the data they have entered. When an unauthorized user opts to view the data, they will see all the data, in two formats:

1 – The unencrypted data will be visible in its normal form.
2 – The encrypted data will be visible in a cipher text form.

When an authorized user wishes to view their encrypted data, they will be required to submit the key received by them after the encryption was first done. After verifying whether the key matches, it will run the decryption algorithm and show the user the details of the data that was previously visible in cipher form.

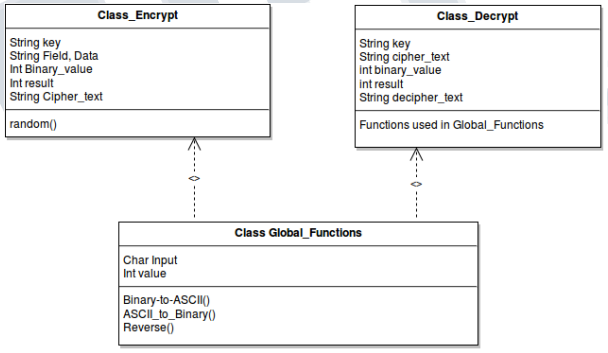



***Fig. 3: Class Diagram for Proposed System***

The above diagram represents the classes required for this application. There will be one global class which will specify functions common to both, the encryption as well as the decryption classes.

The encryption class will also make use of a randomizer function to generate a random key for the encryption process. The decryption class, on the other hand, only utilizes the functions already specified in the global class.

***System Requirements***

***Hardware:***

- Pentium IV or higher
- 512MB RAM or higher
- Internet Connection 256Kbps or higher

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 3, Issue 12, December 2016

***Software:***

***Operating System***:

- Windows 7 or higher OR
- Linux distros with DEB package format( example Ubuntu, Mint )

***Languages and other Software***:

- Java
- Browser to run application
- Struts Framework
- MongoDB Database

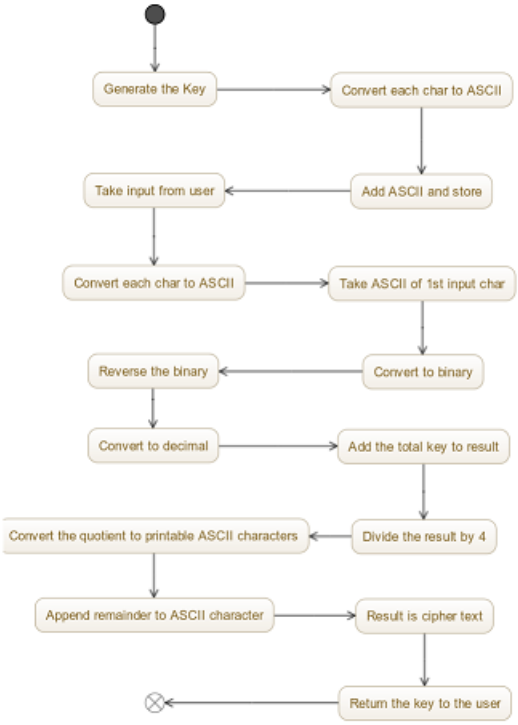
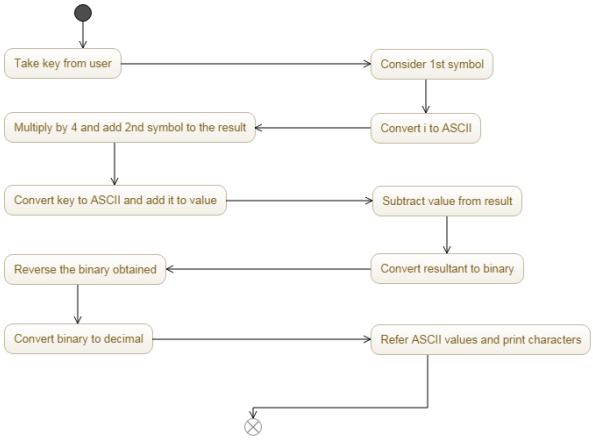



***Fig. 4: Activity Diagram for Proposed System***

The above diagram represents the control flow of the system, and represents the working of the system from start to finish.




## VII. CONCLUSION

As we have come to realize, security plays an important part in the access of data today. Using the appropriate algorithm to protect our data can be a daunting task, as there are several of them, each with their own merits as well as demerits. The REA Algorithm proves to be an efficient algorithm, finding a balance between security as well as efficiency, increasing the speed required to access data as well as providing enough security to protect the data from potential attackers. The REA Algorithm can also be modified to increase its security. While it is not the most secure algorithm in the world, it is fast enough to trade off its security for performance.

Using our proposed system, businesses, small and large, can encrypt data as per their requirements, allowing for greater flexibility in the access of data. Corporations can use this system to automate the process of manually encrypting the data that they store of the individuals who use their services. Also, the users themselves can have the assurance that their data will remain safe and private, only visible to them. Therefore, we can conclude that our system will be beneficial to businesses as well as individuals, and has a scope of endless possibilities.

In the future, we are interested in enhancing the scope of this application to include encryption for multimedia such as images, audio and video.

## REFERENCES

[1] Rajdeep Bhanot and Rahul Hans, "A Review and Comparative Analysis of Various Encryption Algorithms", International Journal of Security and its Applications Vol. 9. No. 4(2015), pp. 289-306, http://dx.dol.org/10.14257/ijsia.2015.9.4.27

[2] Rohini A. Chirde and Prof. S. S. Kulkarni, "Implementing REA Algorithm to Increase Performance of the Encrypted Databases While Query Processing", International Journal of Computer Science and Information Technologies, Vol. 5 (5), 2014, 6556-6561.

[3] J. Daemen and V. Rijmen, Rijndael, "The Advanced Encryption Standard(AES)", Dr. Dobb's Journal, vol. 26, no. 3, pp. 137-139, Mar. 2001.

[4] G. Davida, D. L. Wells, and J. B. Kam, "A database Encryption Systemwith subkeys", ACM Transactions on Database Systems, vol. 6, no. 2, pp. 312-328, 1981.

[5] E. Damiani, S. D. C. D. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Metadata Management in Outsourced Encrypted Databases", in The Second VLDB Workshop on Secure Data.

[6] M. R. Doomun and K. M. S. Soyjaudah, "Analytical Comparison of Crypto-graphic Techniques for Resource Constrained Wireless Security", International Journal of Network Security, vol. 9, no. 1, pp. 82-94, 2009.

[7] D. S. A. Elminaam, H. M. A. Kader, and M. M. Science, Menoufia University, Egypt in 2008. He Hadhoud, "Evaluating the Performance of Symmetric Encryption Algorithms," International Journal of Network Security, vol. 10, no. 3, pp. 213-219, 2010.

[8] S. Lee, T. Park, D. Lee, T. Nam, and S. Kim, "Chaotic Order Preserving Encryption for Efficient and Secure Queries on Databases", IEICE Transactions on Information and Systems, vol. 92, pp. 207 217, 2009.