Jumbling- Salting: An Improvised Approach for Password Encryption

Prathamesh P. Churi
Masters in Information Technology
Shah and Anchor Kutchhi Engineering College
Mumbai, India.
prathamesh.churi@gmail.com

Vaishali Ghate, Kranti Ghag
Assistant Professor,
Shah and Anchor Kutchhi Engineering College
Mumbai, India.
vaishalighate@yahoo.co.in, sakec.krantig@gmail.com

Abstract— **This paper presents an improvised approach for plain text password encryption in the server's database. One of the major aspect of password protection issue is to secure it by means of encryption process. In cryptanalysis, a dictionary attacks or brute force attacks are the most common ways of cracking passwords. A new approach for improvising the scheme of password encryption is using the process of Jumbling-Salting (JS). In order to augment the security aspect regarding passwords, we are devising JS algorithm which prevents dictionary and brute force attacks by increasing the length of cipher text in a considerable limit. In this algorithm, the jumbling process selects characters from pre-defined character set and adding them into the plain password using mathematical modulus (%) function; salting comprises of adding a random string into jumbled password. Ultimately AES block is implemented which obtains a fixed length password which is stored in the server's database. Randomized version of JS algorithm ensures that there is increase in time to crack the cipher text password, by forming a highly secured version of encrypted password.**

Keywords- Encryption, encryption , Algorithm , JS, Cryptography, password.

I. INTRODUCTION

According to Bruce Schneider "Security is a process, not a product."[1]. This famous quote is well echoed by the phenomenon that although there exists numerous amount of security techniques for today, none of them can single-handedly address all the security goals of an organization.

Password is being most common authentication technique which provides the claimant access to system resources. Being the simplest form of authentication technique used, the probability of cracking the password using different combinations is considerably high [2]. Although encryption process on passwords provides partial solution to prevent such attacks, there exists "brute force attack" or "dictionary attack" which has proven this statement to be inefficacious. To overcome the problem of securing encrypted password, we are developing JS (Jumbling –Salting) technique which will increase the length of cipher text by jumbling additional characters to the original string of password.

JS algorithm majorly has of two processes viz. jumbling and salting. In the jumbling process, the password undergoes "addition", "selection" and "reverse" sub-processes. Addition process is generates a value required for determining the number of characters to be added to the plain text password. Selection deals with selecting random characters to be added to the password from predefined character set. In general there are umpteen number of character set in the server. Selection of characters from different character set is also made random. Reverse process reverses the output of selection process based on some pre-defined condition. The condition can be implemented by any mathematical techniques like even odd, prime number or condition of divisibility etc. In salting part, random salt is added to the jumbled string of password. Selection of salt is based on timestamp value which is determined when the user creates his account [2]. Finally, jumbled and salted password is given to the predefined AES algorithm procedure.

Randomized algorithms are particularly effective when attacker who deliberately tries to perform dictionary or brute-force attack [2]. It is said that randomness is ubiquitous in cryptography. The processes involved in JS algorithm are randomized; hence we can achieve "Randomness in Security"[2].

II. RELATED WORK

A. AES Algorithm

Rijndael is a fast algorithm that can be implemented easily on simple processors [3]. It primarily consists of the cryptographic processes like substitution and transposition, shift operation, exclusive OR, and addition operations. AES uses repeat cycles. There are 9 (128 bits), 11 (192 bits), or 13(256 bits) cycles.

B. DES Algorithm

DES is a block cipher, with a 64-bit block size and a 56-bit key [3]. DES consists of a 16-round series of substitution and permutation. In each round, data and key bits are shifted, permutated, X-ORed, and sent through, 8 s-boxes, a set of lookup tables that are essential to the DES algorithm.

978-1-4673-7670-9/15/$31.00 ©2015 IEEE

Decryption in AES as well as DES is essentially the same process, performed in reverse.

III. IMPLEMENTATION

A. System Block Diagram

1. Jumbling block:

Jumbling process includes three sub- processes viz. addition, selection, and reverse process. Process array is given to Jumbling block [2]. Jumbling block prepends some characters from character set and jumbling them with the help modulus function .modulus function is a mathematical function which returns remainder of the division process.

- Addition sub-block:

This block generates principle random value "l" and updating the size of a Process array [4]. The original size of process array P [ ] is 'x' which is now updated to (x + l).

The figure 1 illustrates the position of Process array P[ ].

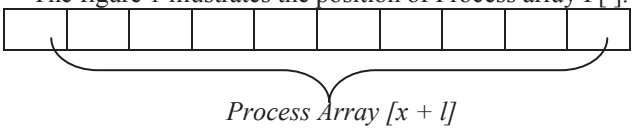

Figure 1. Process Array of JS algorithm

- Selection sub-block:

This block selects random characters from predefined character set say, A. The size of character array is large enough and the character set for a particular password is different. The character set of general Process array is shown in fig. 2 below:

Alphabets	A, B, ..., Y, Z a, b, ..., y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ' ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

Figure 2. Character Set of JS algorithm

- Reverse sub-block:

This block reverses the entire process array based on some predefined mathematical condition. The predefined condition is to check the value of "l" is even or odd. Reverse condition of JS algorithm is to create more confusion in terms of recognizing the actual password. The condition for reverse can be changed which depends upon the application programmer who is developing JS algorithm. for example we can even reverse entire process array if "l" is prime number. The future scope of reversing process array can be done by developing more complex mathematical functions.

2. Salting block

The objective of salting block is to add random string along with jumbled version of password. The criteria of selection of salt is user's sign-up timestamp value. The salt is added in order to make the password more complicated thereby making it difficult for the attacker to crack it.The general form of salt array is shown in fig. 3 below:

Y	Y	Y	Y	M	M	D	D	H	H	m	m	s	s

Where :

Y = Year , M= Month D = date h = Hours in 24 hours format m = minutes s= seconds

Figure 3. Salt Array of JS algorithm

3. AES block:

In this block, we use predefined encryption algorithm AES .AES block is having two sub routines namely, AES encryption an AES decryption.

The block diagram of JS algorithm is shown in fig. 4 below.

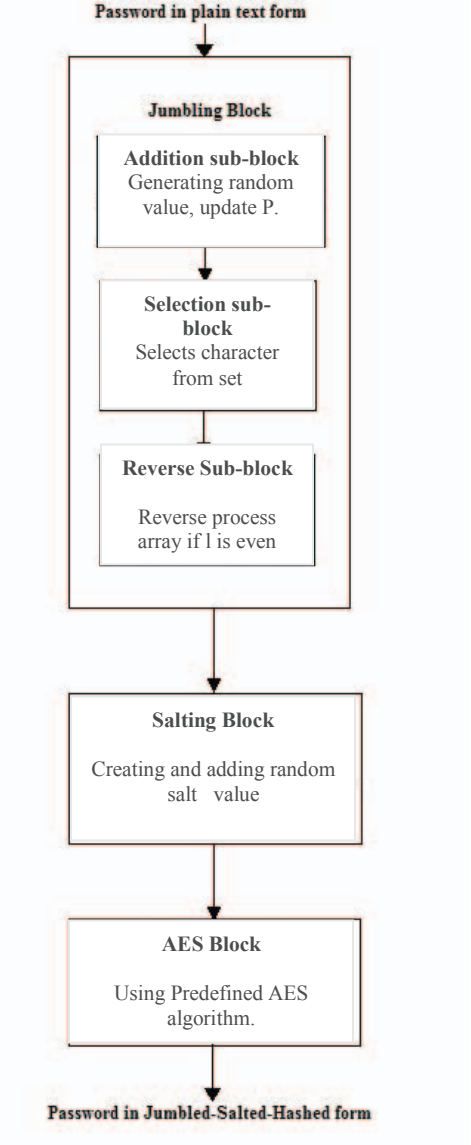

Figure 4. Block diagram of JS algorithm

B. Proposed Algorithm

The pseudo code implementation of JS is given below:

Random ( ): It is a predefined method in most of the programming languages which generates random value from predefined set of values with the valid range.

Process array P [ ]: This array stores the plain-text password with randomly generated characters. We are using this array for encryption as well as decryption process.

Salt array S [ ]: This is use for storing timestamp value from user. In this case, we use user's sign-up time value as timestamp value.

Input: password in plain-text form.

Output: password in Jumbled- salted form.

STEP 1:

INITIALIZE 'x' to 0;

STEP 2:

STORE the length of plain text input password in variable "x";

STEP 3:

CREATE an Process array P [ ] such that P [length =x];

STEP 4:

STORE each character in an array block;

// P {0, 1, 2... x-1) = {characters in process array with index }

/ implementation of Jumbling Process */*

Function jumbling (P [ ])

{

// implementation of jumbling technique: Addition Process

STEP 5:

Label 1: CALL Random () function;

// Random function returns any random value from predefined set of integers within the limit provided by programmer.

SET 'l' as principle random value;

If (l >= x)

STORE random number value in 'l' generated from random () function;

Else

Goto **Label 1**;

Break;

End If

STEP 6:

UPDATE an array P [ ] of size (x + l)

STEP 7:

DEFINE the set of characters A.

//Size (A) = M;

// M = any large value;

//implementation of jumbling technique: Selection Process

/* this process selects characters from given character set. All these symbols are added with plain-text password. These process is also made randomized */

STEP 8:

CALL random ( ) function 'l' times;

// At each iteration, random value is generated by random function which acts as an index of the character in character set.

STEP 9:

FILL the process array with characters

STEP 10:

STORE the original length of an array in variable say "FIX"

For i= 0 to (x+l-1)

While (l! =0)

SET j to 0;

j= (FIX modulus l);

Create 'temp' variable;

Swap (P[i], P[j]);

//output of above modulus function is an index integer within the range 0 to (x + l - 1), Hence we must swap output value with the same index integer with index 0th position value.

l= l-1; // decrement l

End While

End For

// implementation of jumbling technique: Reverse process

STEP 11:

If (l modulus 2 == 0)

REVERSE the entire process array P; // l is EVEN number

Else

Do not REVERSE the process array P; // l is ODD number

End If

Return (P); // pass the jumbled process array to salting function

} // end of jumbling function

/ implementation of Salting Technique */*

Function salting(P[ ])

{

STEP 12:

STORE Timestamp value of Sign-up process for each user;

STEP 13:

OBTAIN the length of timestamp say 't'.

STEP 14:

CREATE Salt [size=t] array which stores random salt characters;

Salt [ ] = {characters obtained from Timestamp value};

STEP 15:

UPDATE an array P of size (x +l + t)

FILL the process array with Jumbled password of size (x + l) and salt characters of size t

Return (P);

// pass the process array to AES function

}

// end of salting function

/ implementation of AES Technique */*

Function AES(P [ ])

{

STEP 16:

Use of predefined AES algorithm can be implemented in AES inbuilt function of any programming language;

}// end of AES function

IV. ANALYSIS PARAMETERS

A. *Plain text size vs. Cipher text size*

In any cryptographic algorithm, it is important to understand the size of the input and the size of output as this is one of the important property of an avalanche effect [5]. Larger the size of the Cipher-text compared with the Plaintext, more secure is the Cipher-text against any brute-force attack. JS, AES and DES cryptosystems uphold this effect and there is no direct relationships between symbols in the Cipher-text to the symbols in the Plaintext.

The table I illustrated below gives the statistical analysis of different set of passwords. For each passwords, the size of cipher text and plain text is calculated.

TABLE I. COMPARISON OF PLAIN TEXT VS. CIPHER TEXT OF JS, AES, DES ALGORITHMS.

TEXT	Plain text size(in bytes)	JS algorithm	AES algorithm	DES algorithm
		Decryption text size (in bytes)		
1	20	128	48	48
2	22	128	48	48
3	28	206	48	192
4	138	400	214	310
5	570	1360	836	1066

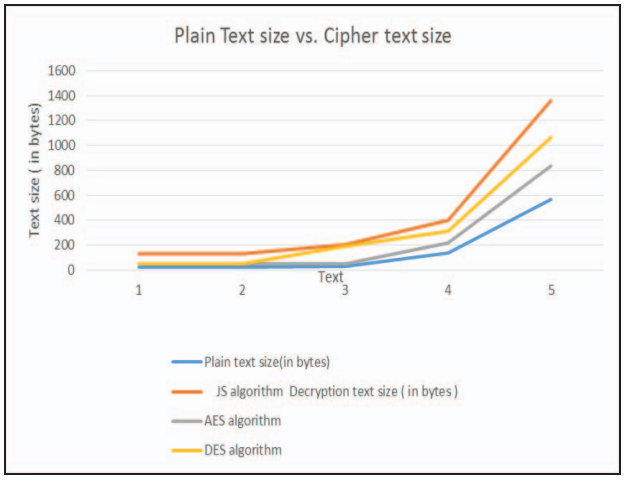

Figure 5. Comparison of plain text Vs. cipher text of JS , AES ,DES algorithms

Fig 5 shows the statistical analysis of Plain text size vs. Cipher text size of JS, AES, and DES algorithm.

Following conclusions can be made for the parameter *Plain text size vs. Cipher text size*:

- In JS algorithm, the size of cipher text as compared to AES and DES algorithms is higher.
- For the larger password string, the cipher text size is drastically increased. The reason behind increasing the size of cipher text of JS algorithm is the number of random characters added in the jumbling block of JS algorithm.
- The range of characters that has been added in the jumbling block has a range from 'x' to '5x' ('x' is the original length of password.). For example, suppose we have the size of plain text password as 8 then the number of characters that has been added in the jumbling block are having range from 8 to 40, which increases the size of cipher text.
- Additionally, the salt (User's sign up timestamp value) has been added in the salting process which increases the size cipher text.

B. *Encryption Time*

The encryption time is the time that an encryption algorithm takes to produce a cipher-text from a plain-text of password [5]. The encryption time has its paramount importance for varied plaintext sizes as this determines the time involved in converting plaintext into cipher-text [5].

JS algorithm has varying encryption time for different set of passwords. As the Jumbling process is randomized, the characters which are to be added in the process array P [ ] has varying length. For example if original password has length 8 character length , then random characters which are to be added might have length 8, 16, 22 etc. depending upon the random number generated during execution .

The password string is taken according to the different set of passwords. [6] It is observed that, the encryption time of different set of passwords is almost same irrespective of the characters present in the password string.

Total Time required for Encryption = Jt + St + AESt
Where,
Jt = Encryption Time for Jumbling process
St = Encryption Time require for Salting process
AESt = Encryption Time require for AES algorithm process

When we compare the encryption time of AES and DES algorithm, the Encryption time is quite less, than JS algorithm. Following table (table II) illustrates the Encryption time of JS, AES and DES algorithm.

TABLE II. ENCRYPTION TIME OF JS, AES, DES ALGORITHMS

Password String	**JS**	**AES**	**DES**
	Encryption Time (in Milliseconds)		
prathamesh	114	81	34
c00lguy123	104	81	40
09o2862o349	119	77	45
PraThamEsH1991	125	80	55
FSt/5ghP*T	115	87	45

The bar chart (fig 6) shows the different encryption times of AES, DES, and JS algorithm.

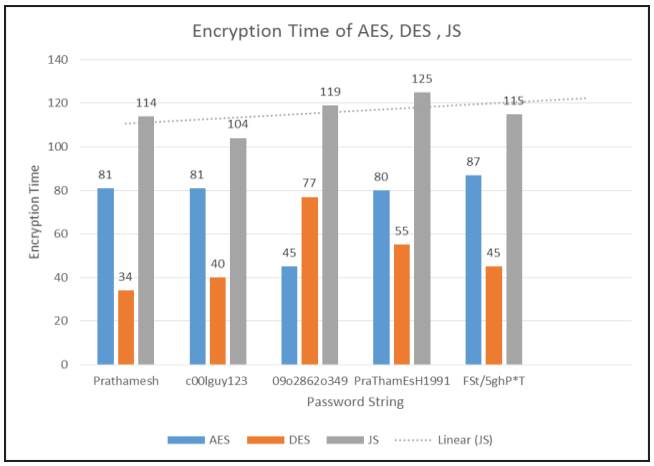

Figure 6. Encryption time of JS , AES , DES algorithms

C. *Decryption Time*

The decryption time is the time that a decryption algorithm takes to reconstruct a plaintext from a cipher-text [5]. The decryption time has its paramount importance for varied cipher-text sizes as this determines the time involved in converting cipher-text back to plaintext [7].

The table III illustrated below has three columns mainly decryption time of JS, AES, DES algorithms.

TABLE III. DECRYPTION TIME OF JS, AES , DES ALGORITHMS

Password String	JS algorithm	AES	DES
	Decryption Time (in Milliseconds)		
prathamesh	70	64	26
c00lguy123	69	60	30
09o2862o349	94	64	30
PraThamEsH1991	95	65	38
FSt/5ghP*T	69	72	26

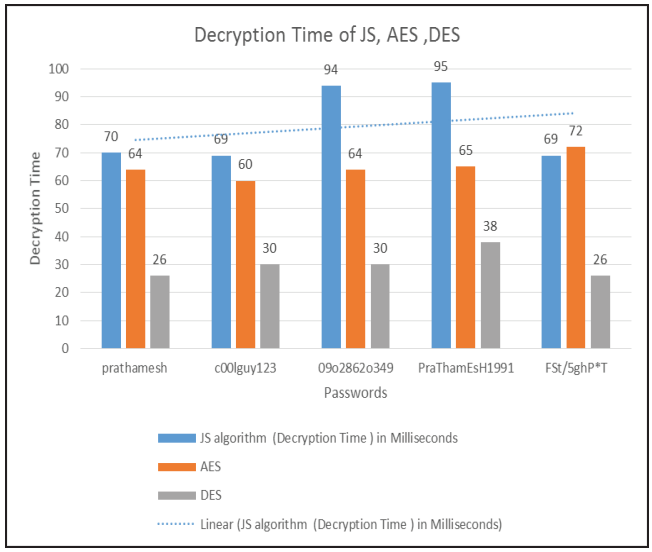

Figure 7. Decryption time of JS , AES , DES algorithms

From, the experimental analysis it is observed that, Encryption as well as decryption time of JS algorithm is quite larger than AES and DES algorithm. There are following reasons which justifies the same:

- The plain text password undergoes jumbling block, where extra characters are added. The total number of characters to be added, will be decided by the random number. For example, if we have plain text password length as 8 (denote as x in the algorithm) then the range of random number generated will be starting from 8 an ending up to 40 (x to 5x).
- In the jumbling process the character will undergo, the addition and selection process where added characters are jumbled with original password string. This process takes $(x + l - 1)$ iterations.
- After jumbling block, algorithm undergoes the process of salting. Where predefined salt is added to the jumbled string of passwords.
- Finally, jumbled and salted version of password undergoes, AES block where the encryption time of AES algorithm is added.

Similarly we can comment the following justification for decryption process as well.

D. *Throughput*

The encryption time can be used to calculate the Encryption Throughput of the algorithms. The decryption time can be used to calculate the Decryption Throughput of the algorithms [5]. Different packet sizes are used in this experiment for JS, AES and DES algorithms.

We have taken the varying text size length in order to calculate throughput in encryption as well as decryption.

TABLE IV. ENCRYPTION TIME OF JS, AES , DES ALGORITHMS FOR THROUGHPUT CALCULATION

Text Size for Encryption (In bytes) (Tp)	Text Size for Decryption (In bytes) (Tp)	Encryption Time		
		JS (Et)	AES (Et)	DES (Et)
10 (A_1)	176	112	99	71
26 (A_2)	222	145	128	111
46 (A_3)	392	185	163	129
570 (A_4)	1378	240	203	167
6808 (A_5)	18568	375	311	269
Tp 7460	Tp 20,610	Et 1077	Et 904	Et 747

TABLE V. DECRYPTION TIME OF JS, AES , DES ALGORITHMS FOR THROUGHPUT CALCULATION

Text Size for Encryption (In bytes) (Tp)	Text Size for Decryption (In bytes) (Tp)	Decryption Time		
		JS (Et)	AES (Et)	DES (Et)
10 (A_1)	176	76	70	48
26 (A_2)	222	90	73	56
46 (A_3)	392	116	101	79
570 (A_4)	1378	118	108	86
6808 (A_5)	18568	168	114	103
Tp 7460	Tp 20,610	Et 589	Et 402	Et 326

The Encryption Throughput and decryption throughput of all three algorithms are tabulated below:

TABLE VI. ENCRYPTION THROUGHPUT OF JS, AES , DES ALGORITHMS

Text	Encryption Throughput of JS algorithm	Encryption Throughput of AES algorithm	Encryption Throughput of DES algorithm
A_1	0.0892 Kb / s	0.1010 Kb / s	0.1408 Kb / s
A_2	0.1793 Kb / s	0.2031 Kb / s	0.2342 Kb / s
A_3	0.2486 Kb / s	0.2822 Kb / s	0.3565 Kb / s
A_4	2.3750 Kb / s	2.8078 Kb / s	3.4131 Kb / s
A_5	18.1546 Kb / s	21.8900 Kb / s	25.3085 Kb / s

TABLE VII. DECRYPTION THROUGHPUT OF JS, AES , DES ALGORITHMS

Text	Decryption Throughput of JS algorithm	Decryption Throughput of AES algorithm	Decryption Throughput of DES algorithm
A_1	2.3157 Kb / s	2.5142 Kb / s	3.0567 Kb / s
A_2	2.4667 Kb / s	3.0410 Kb / s	3.9642 Kb / s
A_3	3.3793 Kb / s	3.8811 Kb / s	4.3280 Kb / s
A_4	11.6779 Kb / s	12.7590 Kb / s	16.0235 Kb / s
A_5	110.5230Kb / s	162.8771Kb/ s	180.271 Kb / s

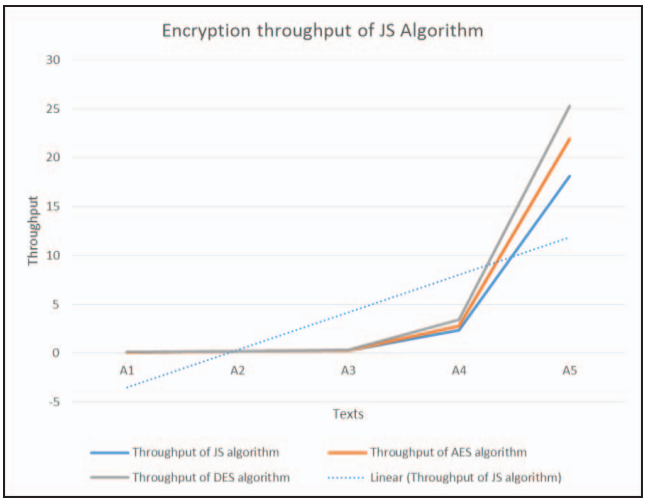

Figure 8. Encryption throughput of JS , AES , DES algorithms

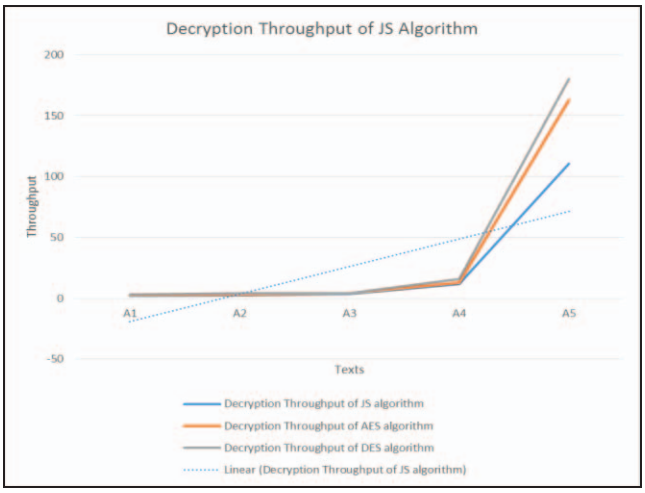

Figure 9. Decryption throughput of JS , AES , DES algorithms

Following conclusions can be made from varying text size cases of JS, AES and DES Algorithms.

- The Encryption and decryption throughput response of JS algorithm is linear.When the size of texts increases the throughput is increases.
- As compared to AES and DES Algorithms, the throughput of JS algorithm is lower as the overhead of Jumbling process increases the Encryption and decryption time of JS algorithm.
- As far as Password of a particular string is concerned, there is no drastic change in the throughputs of JS, AES and DES Algorithms. Hence JS algorithm is prove to be efficient algorithm.

V. CONCLUSION

JS algorithm builds an encrypted version of password which is almost difficult to crack, due to involvement of different randomization processes.

The size of cipher text created in JS algorithm is larger than AES and DES Algorithms. For example, the plain text having length 28 bytes can generate 206 bytes of cipher text length in JS algorithm. With the same size of plain text, AES produces 48 bytes and DES algorithm produces 192 bytes of cipher text .The reason for increasing the larger text size is the extra overhead of the characters which helps in making password difficult to crack.

The encryption time and decryption time of JS algorithm is quite larger than AES and DES algorithms. The reason behind increasing the value of processing time is additional overhead of jumbling and salting process. The jumbling and salting process exhibit the feature of randomization. The average difference between the encryption time of JS algorithm with respect to AES and DES algorithm is 40 and 60 milliseconds respectively. Similarly the average difference between decryption times of JS algorithm with respect to AES and DES algorithm is 10 and 60 milliseconds respectively.

The average encryption throughput of JS algorithm for TEXT SET A (A_1 to A_5) is 4.20884 Kb / s whereas in AES algorithm – it is 5.05682 Kb / s. whereas in DES algorithm it is -5.88924 Kb / s. Similarly the average decryption throughput for TEXT SET A is 26.06916 Kb / s whereas in AES algorithm – it is 37.0012 Kb / s. whereas in DES algorithm it is - 42.4712 Kb / s. the reason behind increasing encryption and decryption throughput is increase in encryption and decryption time of JS algorithm.

The algorithm can be enhanced by securing the principle random number on server side. We can also modify the processes involved in predefined AES algorithm so that complexity of encryption will be increased. Reduction in encryption and decryption time of the algorithm, will further improve the throughput of the algorithm.

ACKNOWLEDGMENT

I am using this opportunity to express my gratitude to everyone who supported me throughout this research. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the research work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project. I am also thankful to Principal Mr. V. C. Kotak and management of my institute for their support and encouragement. I am highly indebted to my Head of Department Ms. Swati Deshpande and guide Ms. Vaishali Ghate and co-guide Ms. Kranti Ghag for their guidance and constant supervision as well as for providing necessary information regarding the research & also for their support in completing the project on research.

I am also thankful to the pioneers of this algorithm Mr. Bhavin save and Ms. Medha Kalelkar for their support. I want to dedicate this work to my brother Mr. Rohan chaudhari for consistently encouraging me throughout this project. Without him my research would have never been published in international level. Last but not least, I want to thank all my Vidyavardhini's College of Engineering and Technology- Information Technology department (where I am currently working as a lecturer) for providing me valuable advice at a very right time of my research.

ABOUT AUTHORS

Prathamesh P Churi is currently pursuing his Master's degree in Information Technology from Shah and anchor kutchhi engineering college, University of Mumbai. He has completed his bachelor's degree in Computer Science from Vidyalankar Institute of Technology, University of Mumbai. He is currently working as a lecturer in Vidyavardhini's college of engineering and technology, University of Mumbai.

Vaishali Ghate is currently working as Assistant professor in Shah and anchor kutchhi engineering college, University of Mumbai. She has 15 years of Experience in teaching. Her area of interests are Wireless Technology and Information Security.

Kranti Ghag is currently working as Assistant professor in Shah and anchor kutchhi engineering college, University of Mumbai. She has 10 years of Experience in teaching. She is pursuing PhD in Computer Science from NMIMS MPSTME, Mumbai, India. Her area of interest are sentimental analysis and data mining.

REFERENCES

[1] Applied Cryptography-Protocols, Algorithms and Source Code in C , John Wiley Publications, ISBN 978-04711-17094)

[2] Prathamesh Churi, Medha Kalelkar, and Bhavin Save, "JSH Algorithm: A Password Encryption Technique using Jumbling-Salting-Hashing", International Journal of Computer Applications (0975-8887), Vol. 92-No.02, April 2014.

[3] William Stallings. Cryptography and Network Security: Principles and Practices, Fourth Edition. Beijing: Electronic Industry Press. 2007: 229~250.

[4] O. Goldreich, S. Goldwasser, and S. Micali, "How to Construct Random Functions", J. ACM, Vol. 33, No. 4, 1986, pp 210-217.

[5] "A Performance Comparison of Data Encryption Algorithms," IEEE Information and Communication Technologies, 2005. ICICT 2005. First International Conference, 2006-02-27, PP. 84- 89.

[6] M. Stamp. Information Security : Principles an practice, Wiley publications , ISBN-13 978-0-471-73848-0

[7] Tingyuan Nie, Chuanwang Songa, Xulong Zhi (2010), "Performance Evaluation of DES and Blowfish Algorithms", Proceedings of 2010 IEEE.