A New Substitution Cipher - Random-X

Falguni Patel, Mohammed Farik

Abstract: Ciphers are the encryption methods to prepare the algorithm for encryption and decryption. The currently known ciphers are not strong enough to protect the data. A new substitution cipher – Random-X, that we introduce in this paper, can be used for password encryption and data encryption. Random-X cipher is a unique substitution cipher which replaces the units of plaintext with triplets of letters. The beauty of this cipher is that the encrypted string of the same plain text is not always same. This makes it strong and difficult to crack. This paper covers the principle, the implementation ideas and testing of Random-X cipher.

Index Terms: Decryption, Encryption, Random-X cipher, Substitution cipher

1 Introduction

There are various types of authentication processes being used for system authentication commonly known as something you know, are or has. The most common among them is some secret which nobody else knows. This is called a password. The system must store the password in a table of a database or file to authenticate it later. Storing the password in plaintext is a bad idea because tables or files are vulnerable to theft. If someone gets the access to this table or file, he/she can find the password. This increases the necessity of Password Encryption If the password is stored encrypted then the system will encrypt this password again and then try to match it with existing encrypted value which will never match. The currently available ciphers and encryption methods do not provide high security. Random-X password encryption method discussed in this paper, is difficult to decrypt or crack. In this paper, briefly explain encryption and substitution ciphers, before detailing Random-X cipher, and its encryption method for implementation, testing, and conclusions.

2 Encryption

Encryption is an interesting piece of technology that works by scrambling data so it is unreadable by unintended parties [2]. The popular ciphers algorithm are RSA (Asymmetric Key Encryption / Public Key Encryption), AES (Symmetric Key Encryption) etc.

2.1 RSA

RSA is a public key encryption algorithm. In this a public key is used to encrypt and a private key is given to decrypt it. Unlike Triple DES, RSA is considered an asymmetric algorithm because of its pair of keys for encryption and decryption. It takes more time and processing power because the result of RSA encryption is a huge mumbo jumbo [2]. This is widely used for Key exchange and Digital signatures.

- *Falguni Patel is a Project Manager - Enterprise Resource Planning (ERP) at Motibhai Group of Companies Ltd - Fiji Islands and a student of Post-Graduate Diploma in Information Technology at UniFiji, Sawenti Campus. Ph: + 679 6668941. Email: falguni_divyesh@yahoo.co.in or falguni.divyesh@gmail.com.*
- *Mohammed Farik (Member, IEEE) is a Lecturer in Information Technology at The University of Fiji. E-mail: mohammedf@unifiji.ac.fj*

2.2 AES

AES (Advanced Encryption Standard) is a symmetric encryption algorithm. It supports both hardware and software implementation [4]. AES is a three block ciphers. Each block Encrypts and decrypts the data in block of 128 bits.

3 Substitution Ciphers

Substitution ciphers is a method of encoding where each characters in the plain text is replaced with a character or a symbol or a string of characters and symbols. Monoalphabetic substitution cipher, ADFG(V)X, Autokey cipher, Alberti cipher, Caesar Cipher, Enigma ciphers, Four-square cipher, Freemason cipher, Kamasutra cipher, Larrabee cipher, Pollux cipher, Polybius cipher, takes a letter of an alphabet and substitutes it with another letter or number. This can be easily cracked with frequency analysis [6].

3.1 Random-X Cipher

The Random-X Cipher gives different encrypted values for one Input string. Though it gives multiple encrypted values, it does not use any key to determine this. This increases the time and power for attacker to crack the algorithm. With Random-X method you can increase the strength by increasing substitution options, for instance Random-3, Random-5, and etcetera. This is difficult to crack with Brute force attack. This idea is based on a multiple character substitution. For each valid password characters (ideally A-Z, a-z, 0-9, and special characters - ! @ # $ % _ * etc.) ASCII value must be identified first. We do not need to store this table into database. We can always call system built-in functions to get ASCII value during application development.

TABLE 1

ASCII Values for Common Characters

Char	ASCII	Char	ASCII	Char	ASCII	Char	ASCII
A	65	a	97	0	48	[	91
B	66	b	98	1	49	]	93
C	67	c	99	2	50	{	123
D	68	d	100	3	51	}	125
E	69	e	101	4	52	@	64
F	70	f	102	5	53	!	33
G	71	g	103	6	54	"	34
H	72	h	104	7	55	$	36
I	73	i	105	8	56	%	37
J	74	j	106	9	57	#	35
K	75	k	107			&	38
L	76	l	108			*	42
M	77	m	109			(	40
N	78	n	110			)	41
O	79	o	111				

P	80	p	112				
Q	81	q	113				
R	82	r	114				
S	83	s	115				
T	84	t	116				
U	85	u	117				
V	86	v	118				
W	87	w	119				
X	88	x	120				
Y	89	y	121				
Z	90	z	122				

Now assign 3 unique strings which is 3 characters long to each of the ASCII value identified in above Table I. Below Table II shows an example of assignment of A-Z, a-z and 0-9. The same assignment can be done for all valid special characters as per password policy. Please ensure that these strings are unique. We can use spreadsheet functions to get a set of unique strings.

TABLE 2

ASCII VALUES WITH 3 SUBSTITUTION (RANDOM-3) OPTIONS

		Random-X		
Char	**ASCII Value**	**1**	**2**	**3**
A	65	7TF	R0J	VO5
B	66	2GO	T6E	AF9
C	67	6WM	G2Z	RI3
D	68	3NH	C0N	TJ0
E	69	3VU	Z3X	BX8
F	70	8YC	G1P	TX9
G	71	1AI	F8O	GM5
H	72	1ZG	H9R	NS3
I	73	4CW	P0J	RY5
J	74	7MO	U6R	YR7
K	75	5QD	V4X	OX7
L	76	7XN	D7A	QC7
M	77	1EW	Y1P	TU8
N	78	9TK	E1I	JW6
O	79	5XQ	H0U	AO8
P	80	1NA	O0G	ZP1
Q	81	0VK	Z9O	UI3
R	82	4EK	X7D	ZY4
S	83	0ZV	Q6R	LO0
T	84	8BH	Z3K	ZF4
U	85	7WW	V7I	MD6
V	86	7XD	K1L	BM0
W	87	7RQ	O6D	OW6
X	88	1IS	D6D	TY8
Y	89	9YO	R4V	QU1
Z	90	8MK	S4X	PP5
A	97	3MR	B3Y	VU3
B	98	2QF	T4V	EN8
C	99	9PM	X9P	HC8
D	100	0WG	U0L	IR9
E	101	0MN	Q4U	WL7
F	102	9UQ	D9D	UO2
G	103	8PD	O2U	BU5
H	104	3DR	H2D	UT7
I	105	9EA	Y4H	UK7
J	106	6RB	D1M	HA7
K	107	9ST	I3X	KE0
L	108	9QA	O3K	KO9
M	109	8EH	H5Z	LY1
N	110	7RT	A1G	XW9
O	111	6DC	F0H	VF2
P	112	5WA	O6E	DM7
q	113	8IF	I2G	BZ1
r	114	0VX	P1X	CF0
s	115	7IM	G9T	QV9
t	116	6IZ	E7Q	CQ6
u	117	9GN	L2I	XW3
v	118	5NP	D8O	UW8
w	119	6SM	Z5B	JZ4
x	120	9SS	C9G	WB0
y	121	1NX	U8B	WZ1
z	122	3JT	R4L	SG6
0	48	4UK	G5F	UZ7
1	49	4YB	O9P	RQ0
2	50	7UG	V3K	LS7
3	51	8YA	A7G	HG6
4	52	5PS	M0F	TS5
5	53	2OZ	X5G	SF5
6	54	2AO	J1M	OY6
7	55	2PW	Z9M	TD0
8	56	1JO	W8L	XY4
9	57	7KU	Z7Y	FJ8

Below is the example of how the Password 'Hello' is encrypted using this algorithm. For each character of a string, get the ASCII Value (from Table 1), pick a random number between 1 and 3, read the value (from Table 2) and concatenate this. For example, the password Hello is encrypted as:

H Random No. 2= H9R
e Random No. 1= 0MN
l Random No. l= 9QA
l Random No. 3= KO9
o Random No. 3= VF2

So, the encrypted string is "H9R0MN9QAKO9VF2". When the same password is encrypted again the string can be as below:

H Random No. 1= 1ZG
e Random No. 2 = Q4U
l Random No. 3 = KO9
l Random No. 2 = O3K
o Random No. 1 = 6DC

So, the encrypted string is "1ZGQ4UKO9O3K6DC".

This way password "Hello" can have 5 * 3 = 15 different encrypted values. If we use 5 substitution options, the password "Hello" will have 5 * 5 = 25 different encrypted values. Every time when the password is entered, this encryption algorithm will convert it into an encrypted string. The encrypted string will be stored in a database table or file for application validation purpose. The Decryption process is the reversal process of above Encryption process. Fig.1 shows the password authentication process. According to this process the password authentication is done by comparing the decrypted value of the encrypted password.

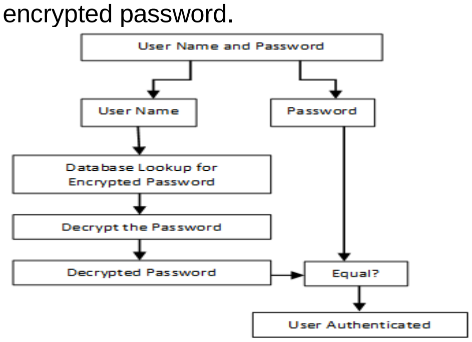

Fig. 1. *Password Authentication Process*

4 Implementation of Encryption Method

To implement this method following four primary functions are required.

4.1 f_encryptChar

The sample function is as shown in Fig.2.

```
using namespace std; // So the program can see cout and endl

/* run this program using the console pauser or add your own getc
int gi_counter = 0;
int gi_prev_Time = 0;
int getMilliCount()
{
    timeb tb;
    ftime(&tb);
    int nCount = tb.millitm + (tb.time & 0xfffff) * 1000;
    return nCount;
}
string f_encryptChar(int ai_ASCII)
{
    gi_counter = gi_counter + 500;
    int li_Time;
    li_Time = getMilliCount() + gi_counter;
    int ai_rand ;
    srand(li_Time);
    ai_rand = (rand()%3) + 1;

    int i = 0;
    string ls_enc_Char;
    //Define Encrypted string for each ASCII Value here
    switch (ai_ASCII)
    {
        case 65: //A
            if (ai_rand == 1)
            {
                ls_enc_Char = "7TF"; return ls_enc_Char;
            }
            else if (ai_rand == 2)
            {
                ls_enc_Char = "R0J"; return ls_enc_Char;
            }
            else if (ai_rand == 3)
            {
```

Fig. 2. *f_encryptChar function*

Arguments: integer (ASCII Value)
Return: String (Encrypted String)
Purpose: This function converts the ASCII value of a character to an encrypted string.

4.2 f_encryptPWD

Argument: String (Password Entered)
Return: String (Encrypted Password)
Purpose: This function converts the Entered Password string to an Encrypted string and returns Encrypted Password. This function calls f_encryptChar() function while looping through each character of the entered password.
The sample figure is as shown in Fig.3.

Fig. 3. *f_encryptPWD function*

```
{
string f_encryptPWD(string as_Pwd_Entered)
{
    string ls_Encrypted_PWD;
    //Looping through the characters of a std::string with an old-fashioned for-loop
    for (std::string::size_type i = 0; i < as_Pwd_Entered.size(); ++i)
    {
        //Read Character
        char ls_char = as_Pwd_Entered[i];
        int li_ASCII_Char = (int)ls_char;
        string ls_Encrypted_Char = f_encryptChar(li_ASCII_Char);
        ls_Encrypted_PWD.append(ls_Encrypted_Char);
    }
    return ls_Encrypted_PWD;
}
int f_decryptChar(string as_EncPWDString)
{
int main(int argc, char** argv) {

    string ls_PWD_Entered;
    //Get the Password from user
    cout<<"Please enter the password to  Test Encrypt: ";
    cin>>ls_PWD_Entered;
    string ls_EncPWD = f_encryptPWD(ls_PWD_Entered);
    cout<<"Encrypted Password is: ";
    cout<<ls_EncPWD<<endl;

    cin.get();
    return 0;
```

4.3 f_decryptChar

Arguments: string (String Encrypted String parse (3chars))
Return: Integer (ASCII Value)
Purpose: This function converts the String parse (3characters) into ASCII Value.
The sample function is as shown in Fig. 4.

```
int f_decryptChar(string as_EncPWDString)
{
    int li_Dec_Ascii;
    if (as_EncPWDString == "7TF")
    { li_Dec_Ascii = 65; return li_Dec_Ascii;}  //A
    else if (as_EncPWDString == "R0J")
    { li_Dec_Ascii = 65; return li_Dec_Ascii;}  //A
    else if (as_EncPWDString == "VO5")
    { li_Dec_Ascii = 65; return li_Dec_Ascii;}  //A
    else if (as_EncPWDString == "2GO")
    { li_Dec_Ascii = 66; return li_Dec_Ascii;}  //B
    else if (as_EncPWDString == "T6E")
    { li_Dec_Ascii = 66; return li_Dec_Ascii;}//B
    else if (as_EncPWDString == "AF9")
    { li_Dec_Ascii = 66; return li_Dec_Ascii;}  //B
    else if (as_EncPWDString == "6WM")
    { li_Dec_Ascii = 67; return li_Dec_Ascii;}  //C
    else if (as_EncPWDString == "G2Z")
    { li_Dec_Ascii = 67; return li_Dec_Ascii;}  //C
```

Fig. 4. *f_decryptChar function*

4.4 f_decryptPWD

Argument: String (Encrypted String)
Return: String (Original Password)
Purpose: This function converts the encrypted password into original password. This function loops through each characters of encrypted string, group into a string of 3 characters and calls f_decryptChar function to decrypt the password. The sample is as shown in Fig. 5.

```
int f_decryptChar(string as_EncPWDString)
{
string f_decryptPWD(string as_EncryptedPWD)
{
    int li_mod = as_EncryptedPWD.length() % 3;
    std::string ls_decryptPWD; string ls_decChar = ""; string ls_readStr = "";
    if  (li_mod != 0 )
    {
        cout<<"The Encrypted Password is not valid.";
        return "Error";
    }
    int li_count = 1; int li_decASCII;
    for (int i = 0; i < as_EncryptedPWD.length(); i++)
    {
        ls_readStr = ls_readStr +  as_EncryptedPWD[i];

        if (li_count == 3 )
        {
            li_decASCII = f_decryptChar(ls_readStr);
            char ls_dec_Char = static_cast<char>(li_decASCII);
            ls_decryptPWD.push_back(ls_dec_Char);
            ls_readStr = "";
            li_count = 0;
        }
        li_count = li_count + 1;
    }
    return ls_decryptPWD;
}
int main(int argc, char** argv) {
```

Fig. 4. *f_decryptChar function*

5 Testing

The C++ program is created to test this algorithm which encrypts and decrypts the password. This program has options to select either Encrypt or Decrypt utility. Sample screen snapshot are as Fig. 5 and Fig. 6.

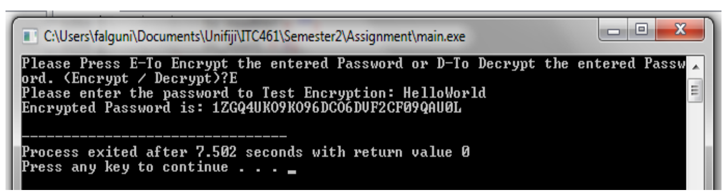

Fig. 5. *Password Encryption Testing*

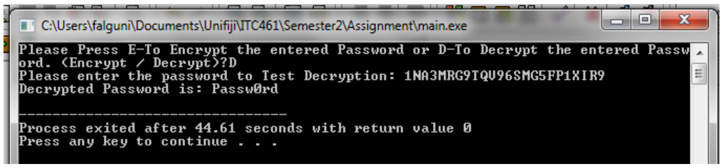

Fig.6. *Password Decryption Testing*

The method suggested here provides strong password protection. As shown in the Test Result Table 3, for same password, the encrypted password is most of the time different. This provides better security. The security can be enhanced by:

- Having 5 character substitution options – Table 2 shows 3 substitution options for each valid character. We can modify this and create 5 options to substitute. With this change, 8 character password will have 8 * 5 = 40 different Encrypted string instead of 8 * 3 = 24.
- Another option to increase the strength is by increasing the length of substitute string from 3 characters to 4 characters. i.e. H = H9R can be changed to MIUT or Li0@ etc.

Table III shows the test results for encryption and decryption algorithm on passwords.

TABLE 3

ENCRYPTION AND DECRYPTION ALGORITHM TESTING

No	Plaintext	Encrypted Password	Decrypted
1	HelloWorld	1ZG0MNKO99QAF0HO6DVF20VXO3KIR9	HelloWorld
2	HelloWorld	1ZGQ4UKO99QA6DCO6DVF20VXO3KU0L	HelloWorld
3	HelloWorld	H9RWL7KO99QAF0HOW66DC0VXO3KIR9	HelloWorld
4	HelloWorld	H9RWL7KO99QAVF27RQ6DCP1XKO90WG	HelloWorld
5	HelloWorld	NS3WL7O3KKO96DCO6DF0HCF09QAU0L	HelloWorld
6	Passw0rd	1NA3MRG9TQV96SMG5FP1XIR9	Passw0rd
7	Passw0rd	ZP13MRG9TQV96SM4UKP1XIR9	Passw0rd
8	Passw0rd	1NAVU3QV97IMZ5BUZ70VX0WG	Passw0rd
9	Passw0rd	ZP13MRG9TQV9JZ44UKP1XIR9	Passw0rd
10	Passw0rd	O0GVU37IMG9TZ5BUZ70VXU0L	Passw0rd
11	RedFlower2	4EK0MNU0LTX9O3KF0HJZ40MNP1XLS7	RedFlower2
12	RedFlower2	4EKQ4UIR98YCO3KF0H6SMQ4UCF07UG	RedFlower2
13	RedFlower2	ZY40MNU0LG1PKO96DCZ5BQ4UCF07UG	RedFlower2
14	RedFlower2	X7DWL7IR98YCO3KVF26SM0MNP1X7UG	RedFlower2
15	RedFlower2	ZY4Q4UU0LTX99QAF0HJZ4WL70VXV3K	RedFlower2

5 CONCLUSION

The cipher method used in this paper provides an alternative and stronger substitution algorithm for password or message security by encryption and decryption.

REFERENCES

[1] Stackoverflow.com. (2016). *Top Hashing and Encryption Algorithms?*. [online] Available at: http://stackoverflow.com/questions/3076222/top-hashing-and-encryption-algorithms [Accessed 9 Oct. 2016].W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[2] Bradford, C. and Bradford, C. (2014). *5 Common Encryption Algorithms and the Unbreakables of the Future*. [online] StorageCraft Technology Corporation. Available at: http://www.storagecraft.com/blog/5-common-encryption-algorithms/ [Accessed 9 Oct. 2016].

[3] VOCAL Technologies, L. (2016). *Triple Data Encryption Standard (Triple-DES)*. [online] Vocal.com. Available at: https://www.vocal.com/cryptography/tdes/ [Accessed 9 Oct. 2016].

[4] SearchSecurity. (2016). *What is Advanced Encryption Standard (AES)? - Definition from WhatIs.com*. [online] Available at: http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard [Accessed 15 Oct. 2016].

[5] Aesencryption.net. (2016). *AES encryption*. [online] Available at: http://aesencryption.net/ [Accessed 15 Oct. 2016].

[6] Cryptool-online.org. (2016). *CrypTool-Online*. [online] Available at: http://www.cryptool-online.org/index.php?option=com_content&view=article&id=47&Itemid=29&lang=en [Accessed 19 Oct. 2016].