© 2015 IJSRSET | Volume 1 | Issue 1 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Asymmetric key encryption based on pseudorandom number generators

Alhussain Amanie Hasn
Peoples' Friendship University of Russia, Moscow
Faculty of Physical, Mathematical and Natural Sciences
Department of Information Technology

ABSTRACT

Cryptography is a fundamental technique for securing information. In this study, has shown how to design asymmetric encryption algorithm based on merging three pseudorandom number generators. The roles of pseudorandom number generators, in the proposed algorithm, serve different roles, one of them is helping to generate a dynamic representation for each character which gives the strength to the encryption algorithm; The proposed encryption algorithm solves the problem of exchange and distribution of private keys over the networks; The various examples and implementation of the algorithm prove that it exchanges the keys and encrypts successfully all the characters, and serves the goals of cryptography.

Keywords: public key, Asymmetric Key Encryption, Pseudorandom Number Generators, encryption, decryption.

I. INTRODUCTION

Cryptography plays an important role in the network security. Cryptography is the science of writing in secret code. The purpose of cryptography is to protect transmitted information from being read and understood by anyone except the intended recipient; so this paper has designed Asymmetric cryptosystem based on pseudorandom number generators to implement and achieve this purpose;

Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using different keys-one is a public key and the other is a private one. It is also known as public-key encryption.[1][2][3][4][5].Asymmetric encryption transforms plaintext into ciphertext using one of the two keys, and the plaintext is recovered from the ciphertext using the other paired one.[6]

The concept of public-key cryptography is evolved from an attempt to attack one of the most difficult problems associated with symmetric encryption, which is key distribution, which requires either (1) that two communicants already share a key, which somehow has been distributed to them; or (2) the use of a key distribution centre[7][8][9][10].

This paper is proposed an algorithm for asymmetric encryption. First, it generates the values of three pseudorandom number generators which are Linear Congruential, Lagged Fibonacci and Blum Blum Shum generators, and three new internal sequences which are derived from the previous sequences by applying mathematical and logical operands. Each of these sequences plays a different role in the proposed encryption algorithm; second, it uses the generated sequences to encrypt the data; the algorithm would generate internally dynamic representation for each character, which depends on the private key.

The rest of the paper is organized as follow: In Section II, the proposed Pseudorandom Number Generators methods is introduced; Section III discusses about the proposed encryption algorithm; Section IV experimental results; Section V concludes the paper.

II. METHODS AND MATERIAL

2.1 Pseudorandom Number Generators

A pseudorandom number generator (PRNG), also known as a deterministic random bit generator (DRBG), is an algorithm for generating a sequence of numbers that approximates the properties of random numbers [7][8]. Because the pseudorandom number generators are periodic and deterministic so this article is supposed an algorithm that is used three generators together (this gives one more level of the security), the methods of PRNG algorithm that are used in this algorithm:

- **Linear Congruential generator**

The generator is defined by the recurrence relation:
Xn+1=(a Xn +c)mod m Where X is the sequence of pseudorandom values, and m –"modulus" a– the "multiplier"; c- the "increment", X0 – the "seed "or “initial value” [3].

- **Lagged Fibonacci generator**

The Fibonacci sequence may be described by the recurrence relation:
Xn+1= (Xn+Xn-1) mod m where m modulus, X0 initial value [3]

- **Blum Blum Shub (B.B.S.) generator**

The generator is defined by the recurrence relation:
Xn+1=Xn2 mod m where m - modulus, X0 - initial value [3].

The algorithm is just required to enter the values of linear congruential generator and it will generate the values of both Fibonacci and Blum Blum Shub generators internally because all of these generators have shared the parameters Xn and m. Based on these three generators would generate three sequences of numbers which each of them plays different roles in the algorithm as shown in fig. 1. The sequences are:

I. ASCII sequence: This is the result of adding both linear and Fibonacci generators. It would be used to generate dynamic representation for each character and is defined by the formula: [(a Xn +c) mod m+ (Xn+Xn-1) mod m] mod m.
II. Cross sequence: this sequence is used to define the position of cross point which would be used in the encryption algorithm, this sequence is created by the following steps:
    a. Convert the values of both linear and Blum Blum Shub generators into binary (16 bit) values.
    b. perform sequentially bit by bit the logical operators (or, xor, and) between the two consecutive values obtained in the previous step;
    c. Take mod m of the previous result, and convert it into decimal representation.

III. Encryption sequence: this sequence is used in the encryption process and is generated by the following steps:
    a. Convert the values of both Fibonacci and blum blum shub generators into binary (16 bit) values.
    b. perform sequentially bit by bit the logical operators (xor, and, or) between the consecutive values obtained in the previous step;
    c. Take mod m of the previous result, and convert it into decimal representation.

(The various generated sequences of numbers, and how they are derived with the various roles that are played in the proposed algorithm give more levels of the security to the algorithm).

Example of the created sequences with the parameters of the private key (initial value=8, modulus=500, multiplier =7, increment=6, size=100) is shown in fig. 2.

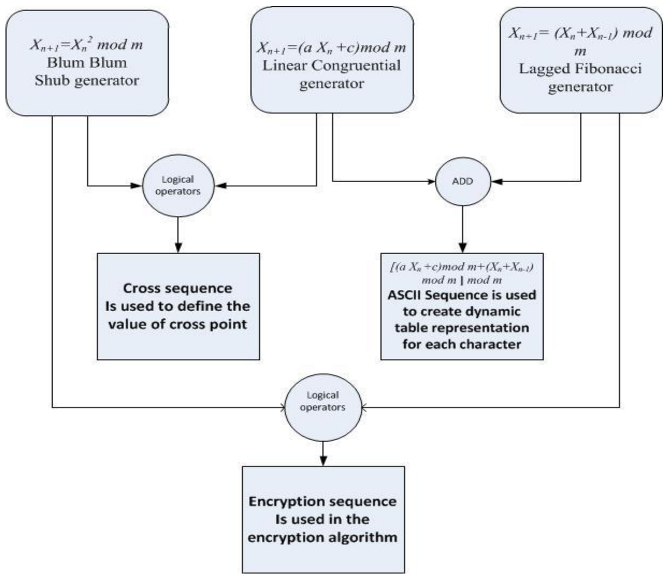

$X_{n+1}=X_n^2 \ mod \ m$
Blum Blum Shub generator
$X_{n+1}=(a\,X_n+c) mod \ m$
Linear Congruential generator
$X_{n+1}=(X_n+X_{n-1}) \ mod \ m$
Lagged Fibonacci generator
Logical operators
ADD
Cross sequence Is used to define the value of cross point
$[(a\,X_n+c) mod \ m+(X_n+X_{n-1}) \ mod \ m] \ mod \ m$
ASCII Sequence is used to create dynamic table representation for each character
Logical operators
Encryption sequence Is used in the encryption algorithm

Figure 1: schema of the different Roles of the proposed generators

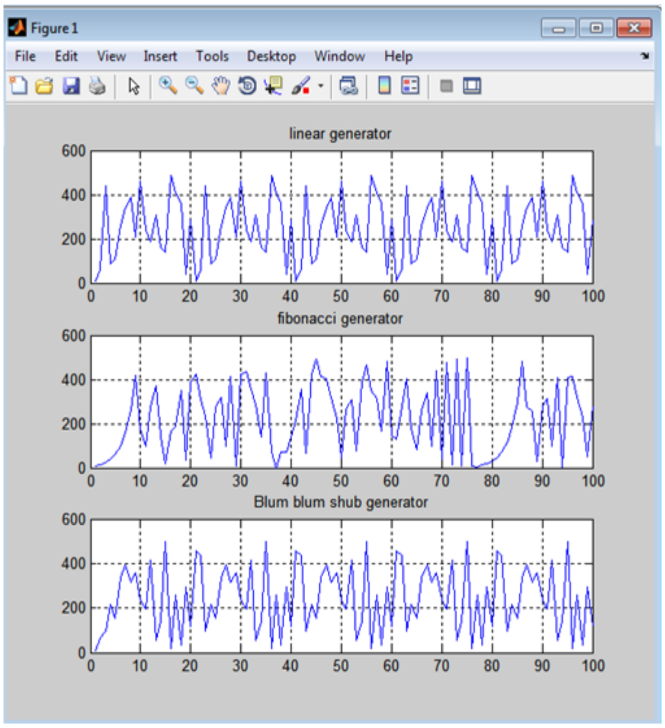


Figure 2: Example of the different values created by the generators which shared the same parameters

2.2 The Proposed Encryption Algorithm:

A. The Secret Key

The secret key is consists of five parameters; which provide strength to the algorithm rendering it difficult for cryptanalysis by intruder (one more level of the security). The five parameters of the secret key are:

Key= {initial value, increment, multiplier, modulus, size}[4]

Initial value, increment, multiplier, modulus are the parameters of Linear Congruential method whose values are known only to the intended sender and recipient. And in the same time (***initial value, modulus***) is used to produce Fibonacci and Blum blum shub sequences of numbers internally in the cryptosystem, after that internal sequences of number (ASCII , Cross and Encryption sequences)are generated.

size represents the size of sequence in which random numbers are stored.

B. Public key

Description of the Algorithm:

The steps to generate public key could be summarized as follow:

1) Generate three random numbers (R_n) in the range 0-15, and convert them into hexadecimal value (HEX (R_n)) $_{16}$.
2) Convert each component of private key into hexadecimal value (HEX (PK_n)) $_{16}$.
3) Apply XOR between hexadecimal representations of both random numbers and private key components:

   (HEX (XOR_n)) $_{16}$ =(HEX (R_n)) $_{16}$ $\oplus$ (HEX (PK_n)) $_{16}$.
4) Convert the values obtained in step3 into binary (12bit) representation (HEX (XOR_n))$_2$ and add the binary (12bit) representation of random numbers (HEX (R_n))$_2$ to the end of numbers.
5) Crossover the two consecutive bytes of data obtained in step4 according to the middle of their sizes i.e. 6bits;
6) Convert the produced bytes obtained in step 5 into Hexadecimal values (HEX (CrossoverBytes))$_{16}$.
7) Public key = sequence of the produced hexadecimal numbers obtained in step 6.

Example of applying the public key algorithm:

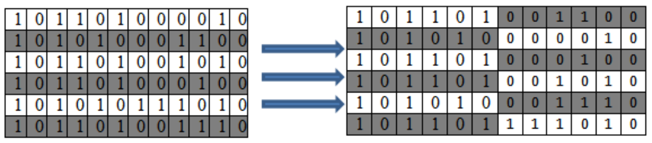
Step 1: Generate three random numbers (Rn) in the range 0-15:
R1= $(11)_{10}$= $(B)_{16}$; R2= $(4)_{10}$= $(4)_{16}$; R3= $(14)_{10}$= $(E)_{16}$;

Step 2: Convert each component of private key into hexadecimal value:
Initial value= $(12)_{10}$=$(C)_{16}$, modulus= $(450)_{10}$= $(1C2)_{16}$,
Multiplier= $(4)_{10}$= $(4)_{16}$, Increment= $(10)_{10}$= $(A)_{16}$, size= $(500)_{10}$= $(1F4)_{16}$

Step 3: XOR between hexadecimal representations of both random numbers and private key components:
$(C)_{16} \oplus (B4E)_{16} = (B42)_{16}$; $(1C2)_{16} \oplus (B4E)_{16} = (A8C)_{16}$;
$(4)_{16} \oplus (B4E)_{16} = (B4A)_{16}$; $(A)_{16} \oplus (B4E)_{16} = (B44)_{16}$;
$(1F4)_{16} \oplus (B4E)_{16} = (ABA)_{16}$;

Step 4: Convert the values obtained in step3 into binary (12bit) representation and add the binary (12bit) representation of random numbers to the end of numbers.
$(B42)_{16}$= $(101101000010)_2$; $(A8C)_{16}$ = $(101010001100)_2$;
$(B4A)_{16}$= $(101101001010)_2$; $(B44)_{16}$ = $(101101000100)_2$;
$(ABA)_{16}$ = $(101010111010)_2$; $(B4E)_{16}$= $(101101001110)_2$

Step 5: Crossover the two consecutive bytes of the data obtained in step4 according to the middle of their sizes:



1	0	1	1	0	1	0	0	0	0	1	0
1	0	1	0	1	0	0	0	1	1	0	0
1	0	1	1	0	1	0	0	1	0	1	0
1	0	1	1	0	1	0	0	0	1	0	0
1	0	1	0	1	0	1	1	1	0	1	0
1	0	1	1	0	1	0	0	1	1	1	0

1	0	1	1	0	1	0	0	1	1	0	0
1	0	1	0	1	0	0	0	0	0	1	0
1	0	1	1	0	1	0	0	0	1	0	0
1	0	1	1	0	1	0	0	1	0	1	0
1	0	1	0	1	0	0	0	1	1	1	0
1	0	1	1	0	1	1	1	1	0	1	0

Step 6: Convert the produced bytes obtained in step 5 into Hexadecimal values
$(101101001100)_2$= $(B4C)_{16}$;($101010000010)_2$= $(A82)_{16}$;
$(101101000100)_2$= $(B44)_{16}$;($101101001010)_2$= $(B4A)_{16}$;
$(101010001110)_2$= $(A8E)_{16}$; ;($101101111010)_2$= $(B7A)_{16}$

Step 7: Public key = sequence of the produced hexadecimal numbers obtained in step6.

Public key=B4C A82 B44 B4A A8E B7A

(Note: the public key for the same private key is not static, but it depends on the generated random numbers this gives one more level of security to the algorithm [3]).

The public key is sent directly from the sender to the receiver, before starting the encryption process by separated secret channel, so the receiver would retrieve the private key from the public key by reverse the steps of obtaining the public key and based on the symmetry of the operations involved and the symmetry of XOR operation.

C. Encryption Process:

Before starting the encryption process; there is a step of preparing and sending the public key to the receiver (this step is done just one time), after that a set of characters with corresponding values are generated based on the addition of the value of ASCII cod to ASCII Sequence (which is discussed earlier) let's call it ASCII table.

Attention should be taken that each character in ASCII table must be unique (more conditions and constrains implemented inside cryptosystem).the benefit here is that the ASCII values would not be static, but they would vary depending on the secret key (this is increase one more level of security);

The encryption process comprises of the following steps:

1. Get the value of first character in plaintext from ASCII table.
2. Convert the previous value into binary (8bits)
3. Apply XOR operation between the obtained value in step2, and the binary representation (8bits) of Encryption Sequence of numbers.
4. Take mode 7 of the Cross sequence to get decimal values ranging from 0 to 6, which would form cross point: Cross Point=mod (Cross Sequence, 7).
5. Divide each XOR result into two parts according to the Cross Point, and convert each part into hexadecimal value.
6. Cipher text would be the sequence of hexadecimal numbers that generated in previous step.

D. Decryption Process:

Before starting the decryption process, if it is the first time, a private key should be retrieved from public key as descried earlier, then the six sequences of numbers are generated, after that ASCII table is created in the same way mentioned in encryption process, finally the steps of decryption are just reversal of the encryption.

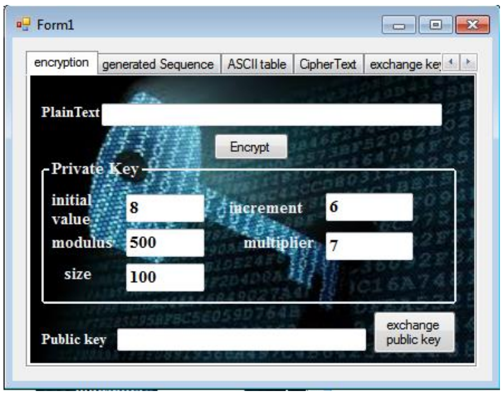
III. RESULTS AND DISCUSSION

A. Exchange the keys

Before starting the encryption process, the generated public key based on the used private key should be sent firstly as mentioned in section 2.2; this gives the cryptosystem the ability to exchange the pair of keys without caring about the problem of key distributions and storing.

- On the Sender side*:*

1) Enter the private key that would be used in encryption. (As shown in Fig.3)



Form1

encryption | generated Sequence | ASCII table | CipherText | exchange ke

PlainText

Encrypt

Private Key

initial value: 8
increment: 6
modulus: 500
multiplier: 7
size: 100

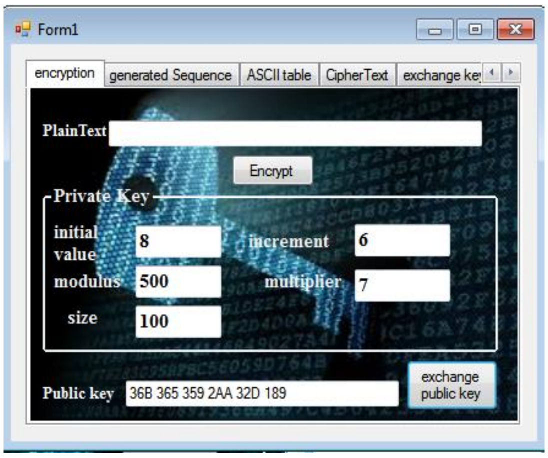
Public key

exchange public key

Figure 3: The private key on the sender side

2) Press the button "exchange public key"

The public key would be calculated on sender side (as shown in Fig4.), and send directly to receiver side (shown in Fig5);

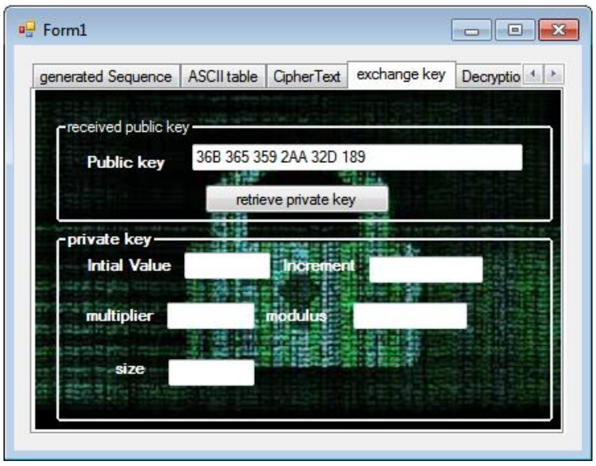

Form1

PlainText	
initial value	8
increment	6
modulus	500
multiplier	7
size	100
Public key	36B 365 359 2AA 32D 189

Figure 4: Calculation of public key on sender side

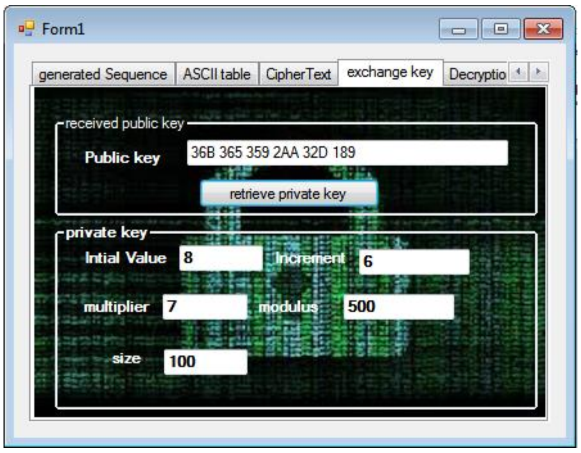
- Receiver side:

After the public key is received (shown in Fig5.), the receiver could calculate the private key from the received public key when pressing the button "retrieve private key", (shown in Fig6).


Form1

Public key	36B 365 359 2AA 32D 189
Intial Value	
Increment	
multiplier	
modulus	
size	

Figure 5: The received public key on the receiver side.


Form1

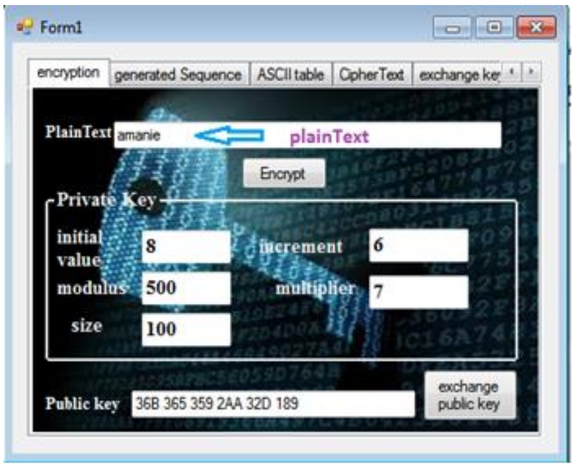
Public key	36B 365 359 2AA 32D 189
Intial Value	8
Increment	6
multiplier	7
modulus	500
size	100

Figure 6: Calculation of the private key from the received public key on the receiver side

B. Encryption

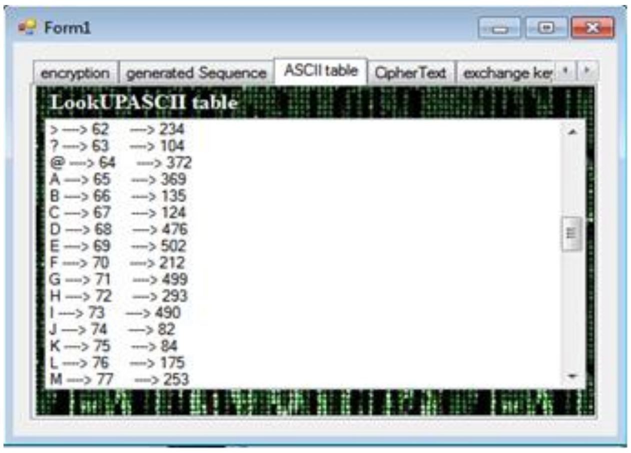
Plaintext: "amanie" If the parameters of private key are: initial value=8, modulus=500, multiplier =7, increment=6, size=100 (as shown in Fig7) is chosen, the dynamic generated "ASCII table" which is based on the private key is shown in Fig.8.

The ciphertext would be= "0803 2800 1705 0700 0119 0F01" as shown in Fig9; In decryption process we will obtain the plaintext= "amanie" as shown in Fig10;


Form1

PlainText	amanie
initial value	8
increment	6
modulus	500
multiplier	7
size	100
Public key	36B 365 359 2AA 32D 189

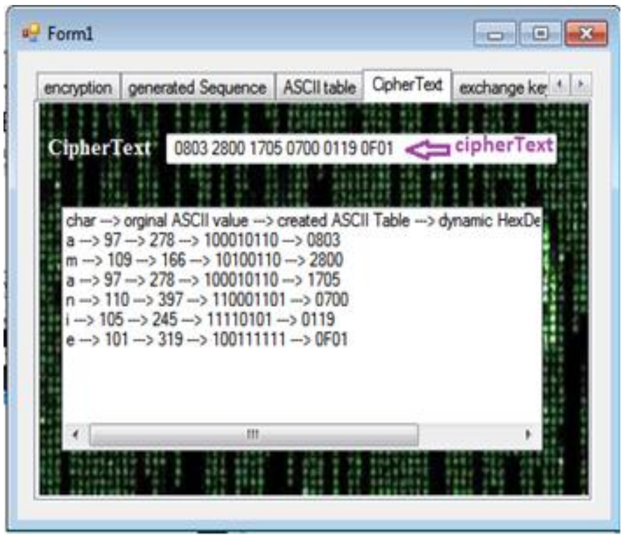
Figure7: Plain text and private key



LookUPASCII table

```
> —> 62    —> 234
? —> 63    —> 104
@ —> 64    —> 372
A —> 65    —> 369
B —> 66    —> 135
C —> 67    —> 124
D —> 68    —> 476
E —> 69    —> 502
F —> 70    —> 212
G —> 71    —> 499
H —> 72    —> 293
I —> 73    —> 490
J —> 74    —> 82
K —> 75    —> 84
L —> 76    —> 175
M —> 77    —> 253
```

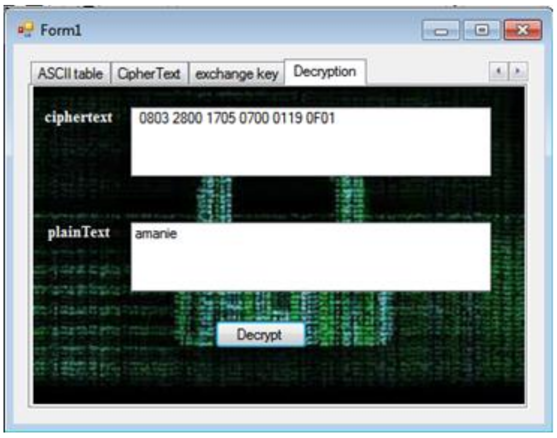
Figure 8: The generated ASCII table



CipherText 0803 2800 1705 0700 0119 0F01

```
char --> orginal ASCII value --> created ASCII Table --> dynamic HexDe
a --> 97 --> 278 --> 100010110 --> 0803
m --> 109 --> 166 --> 10100110 --> 2800
a --> 97 --> 278 --> 100010110 --> 1705
n --> 110 --> 397 --> 110001101 --> 0700
i --> 105 --> 245 --> 11110101 --> 0119
e --> 101 --> 319 --> 100111111 --> 0F01
```

Figure 9: Ciphertext.



ciphertext 0803 2800 1705 0700 0119 0F01

plainText amanie

Figure 10. The produced plaintext on receiver side

IV. CONCLUSION

In conclusion, we feel that FSO technology aided by suitable channel coding techniques can be used to realize reliable high speed communication short distance links across distances less than four kilometres under good as well as adverse weather conditions with the level of data integrity being as good as is required by the application. This development can help in widespread deployment of this attractive technology with its benefits of enabling high data-rate communication with the advantages of quick deployment time, high security and no spectral licensing requirements.

V. REFERENCES

[1] Bayaki, E.; Schober, R., "Performance and Design of Coherent and Differential Space-Time Coded FSO Systems," Lightwave Technology, Journal of , vol.30, no.11, pp.1569,1577, June1, 2012

[2] Safari, M.; Uysal, M., "Do We Really Need OSTBCs for Free-Space Optical Communication with Direct Detection?," Wireless Communications, IEEE Transactions on , vol.7, no.11, pp.4445,4448, November 2008

[3] Islam, M.S.; Majumder, S.P., "Performance analysis of a free space optical link using Alamouti type Space Time Block Code with weak turbulent condition," Electrical & Computer Engineering (ICECE), 2012 7th International Conference on , vol., no., pp.287,290, 20-22 Dec. 2012

[4] Etty J. Lee and Vincent W. S. Chan, Part 1: Optical Communication Over the Clear Turbulent Atmospheric Channel Using Diversity.

[5] Fang Xu, Ali Khalighi, Patrice Causs´e, Salah Bourennane: Channel coding and time-diversity for optical wireless links.

[6] X. Zhu and J. M. Kahn. “Free-Space Optical Communication Through Atmospheric Turbulence Channels,” IEEE Trans. Commun., 50:1293–1300, August 2002.

[7] D. Kedar and S. Arnon, “Urban Optical Wireless Communication Networks: The Main Challenges and Possible Solutions.” IEEE Commun. Mag., 42:S2–S7, May 2004.

[8] L. C. Andrews and R. L. Phillips, Laser Beam Propagation through Random Media (SPIE Press, Bellingham,Washington, 2005), 2nd ed.

[9] Murat Uysal, Jing (Tiffany) Li and Meng Yu, “Error Rate Performance Analysis of Coded Free-Space Optical Links over Gamma-Gamma Atmospheric Turbulence Channels,” IEEE Transactions on Wireless Communication, Vol. 5, NO. 6, June 2006