DEEPIKARANI K, D.S BHAVANI, K. RAVI KISHORE, D.DEEPIKA/ International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622
www.ijera.com Vol. 2, Issue 1,Jan-Feb 2012, pp.672-677

DATA TRANSMISSION USING TCP, GZIP & TINY ALGORITHMS

DEEPIKARANI K*, D.S BHAVANI, K. RAVI KISHORE***, D.DEEPIKA******

*ASSISTANT PROFESSOR, HITMA, HYDERABAD
**M.TECH STUDENT JNTUH, HYDERABAD
***SENIOR PROJECT ENGINEER, NETWORK SECURITY, CDAC, HYDERABAD
****ASSISTANT PROFESSOR, MGIT, HYDERABAD

Abstract -- ***The aim of the Paper is to provide the security during transmission of the data. Commonly used technologies are cryptography, Compression and decompression. This can be used for secure and fast sending for security purpose we are using the Cryptography technology and to increase the file transfer speed we are using the compression and decompression technologies. In this paper we described lot of processes for secure data transmission. First perform the compression and decompression techniques for decrease the file size. It will increase the transmission speed here maintains compression and decompression technique based on the GZIP Algorithm. Than perform the cryptography technique based on the TINY Encryption Algorithm. In sending process the TCP (Transaction control protocol) was involved.***
Keywords – TINY, GZIP, TCP, cryptography, compression, decompression,

INTRODUCTION

System definition is the process of obtaining a clear understanding of the problem space such as your business opportunities; user needs, or market environment and defining an application or system to solve that problem. In the existing system, file transfer is not a secured transaction. User Profiles and access controls are not integrated to provide higher-level security in data transfer. Encryption and decryption implementation through a character user interface is a complicated process where the user or the administrator is to follow some complex process.

The information generated at the client side while under the standard of transfer should be secure enough and protected by any intrusions and interceptions that may occur while the information is transferred. The overall system should concentrate on the best algorithm that can be implemented for al the resource standards that can be implemented as per the standards of the technical quality. In the traditional 2-tier architecture there existed only the server and the client. In most cases the server was only a data base server that can only offer data. Therefore majority of the business logic i.e., validations etc. had to be placed on the clients system. This makes maintenance expensive. Such clients are called as ‘fat clients’. This also means that every client has to be trained as to how to use the application and even the security in the communication is also the factor to be considered.

Since the actual processing of the data takes place on the remote client the data has to be transported over the network, which requires asecured format of the transfer method. How to conduct transactions is to be controlled by the client and advanced techniques implementing the cryptographic standards in the executing the data transfer transactions. Present day transactions are considered to be "un-trusted" in terms of security, i.e. they are relatively easy to be hacked. Nevertheless, sensitive data transfer is to be carried out even if there is lack of an alternative. Network security in the existing system is the motivation factor for a new system with higher-level security standards for the information exchange to provide ease and secured file maintenance and management in a distributed environment.

So in this paper we are going to propose that the system should have the following features : The transactions should take place in a secured format between various clients in the network. The validation code should be placed on the server and not on the client such that file transfer takes place between only the registered users only. This leads to a thin client, which is more desirable. The server should identify the type of request (GET/POST), file access permissions and perform appropriate action. It should also identify the user and provide the communication according to the prescribed level of security with transfer of the file requested and run the required process at the server if necessary. When responding to the client, the server should send necessary information such as User authorization and authentication information, Encryption, Decryption types and their level of hierarchy etc.

RELATED WORK

In this paper we are going to focus on this following techniques in-order to send the data securely using cryptography.

Compress and Decompress Technique:

This technique maps arbitrary input into printable character output. The form of encoding has the following relevant characteristics. The range of the function is a character set that is universally re-presentable at all sites, not a specific binary encoding of that character set. Thus, the characters themselves can be encoded into whatever form is needed by a specific system. For instance, the character 'E' is represented in ASCII system as a hexadecimal 45 and in EDCDIC- based system as hexadecimal- c5.The character set consists of 65 printable characters, one of which is used for padding. With 2^6 = 64 available characters, each character can be used to represent 6 bits of input.

No control characters are included in the set. Thus, the message encoded in Radix-64 can traverse mail-handling system. That scans the data stream for control characters. The hyphen character "- " is not included.

Cryptography Technique (Tiny encryption):

The Tiny Encryption Algorithm (TEA) is a cryptographic algorithm designed to minimize memory footprint and maximize speed. It is a Feistel type cipher that uses operations from mixed (orthogonal) algebraic groups. This research presents the cryptanalysis of the Tiny Encryption Algorithm. In this research we inspected the most common methods in the cryptanalysis of a block cipher algorithm. TEA seems to be highly resistant to differential cryptanalysis, and achieves complete diffusion (where a one bit difference in the plaintext will cause approximately 32 bit differences in the cipher text) after only six rounds. Time performance on a modern desktop computer or workstation is very impressive.

As computer systems become more pervasive and complex, security is increasingly important. Cryptographic algorithms and protocols constitute the central component of systems that protect network transmissions and store data. The security of such systems greatly depends on the methods used to manage, establish, and distribute the keys employed by the cryptographic techniques. Even if a cryptographic algorithm is ideal in both theory and implementation, the strength of the algorithm will be rendered useless if the relevant keys are poorly managed.

The following notation is necessary for our discussion.

• Hexadecimal numbers will be subscripted with "*h*," e.g., 10 = 16. *h*

Bitwise Shifts: The logical shift of x by y bits is denoted by $x << y$. The logical right shift of x by y bits is denoted by $x >> y$.

Bitwise Rotations: A left rotation of x by y bits is denoted by $x <<< y$. A right rotation of x by y bits is denoted by $x >>> y$.

Exclusive-OR: The operation of addition of n-tuples over the field (also known as 2F exclusive-or) is denoted by $x \oplus y$.

The Tiny Encryption Algorithm is a Feistel type cipher that uses operations from mixed (orthogonal) algebraic groups. A dual shift causes all bits of the data and key to be mixed repeatedly. The key schedule algorithm is simple; the 128-bit key K is split into four 32-bit blocks K = (K[0], K[1], K[2], K[3]). TEA seems to be highly resistant to differential cryptanalysis and achieves complete diffusion (where a one bit difference in the plaintext will cause approximately 32 bit differences in the cipher text). Time performance on a workstation is very impressive.

Block ciphers where the cipher text is calculated from the plain text by repeated application of the same transformation or round function. In a Feistel cipher, the text being encrypted is split into two halves. The round function, F, is applied to one half using a sub key and the output of F is (exclusive-or-ed (XORed)) with the other half. The two halves are then swapped. Each round follows the same pattern except for the last round where there is often no swap. The focus of this thesis is the TEA Feistel Cipher.

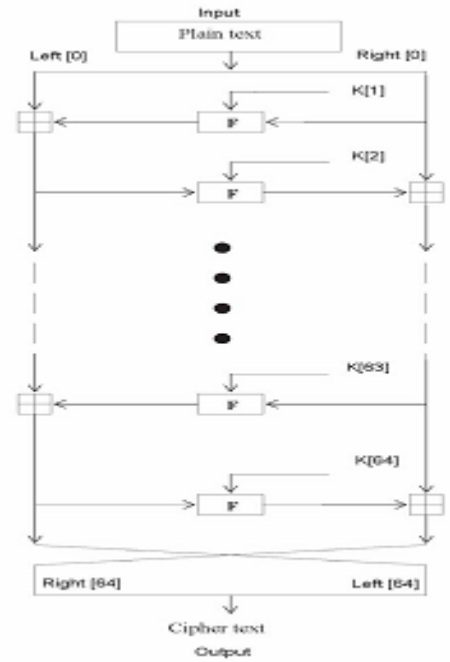

Fig.1: TEA Feistel Cipher

The inputs to the encryption algorithm are a plaintext block and a key K .The plaintext is P = (Left[0], Right[0]) and the cipher text is C =

(Left[64], Right[64]). The plaintext block is split into two halves, Left[0] and Right[0]. Each half is used to encrypt the other half over 64 rounds of processing and then combine to produce the cipher text block.

• Each round *i* has inputs Left[*i*-1] and Right[*i*-1], derived from the previous round, as well as a sub key K[*i*] derived from the 128 bit overall K.

• The sub keys K[*i*] are different from K and from each other.

• The constant delta =$(5^{1/2}-1)*2^{31}$ =9E3779B h , is derived from the golden number ratio to ensure that the sub keys are distinct and its precise value has no cryptographic significance.

• The round function differs slightly from a classical Fiestel cipher structure in that integer addition modulo 2^{32} is used instead of exclusive-or as the combining operator.

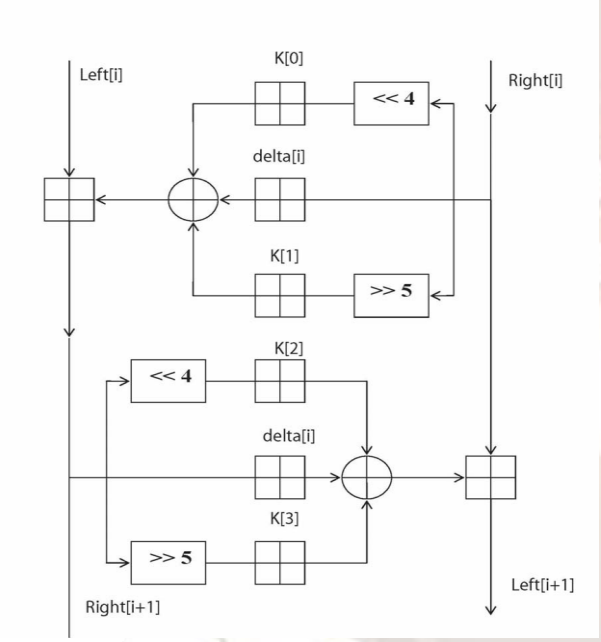

Fig.2: Internal details of TEA

Above Fig.2 presents the internal details of the *i*th cycle of TEA. The round function, F, consists of the key addition, bitwise XOR and left and right shift operation. We can describe the output (Left[*i* +1] , Right[*i* +1]) of the *i*th cycle of TEA with the input (Left[*i*] ,Right[*i*]) as follows

Left [*i*+1] = Left[*i*] F (Right[*i*], K [0, 1], delta[*i*]),

Right [*i* +1] = Right[*i*] F (Right[*i* +1], K [2, 3], delta[*i*]),

delta[*i*] = (*i* +1)/2 * delta,

The round function, F, is defined by

F(M, K[*j*,*k*], delta[*i*]) = ((M << 4) K[*j*]) ⊕ (M delta[*i*]) ⊕ ((M >> 5) K[*k*]).

The round function has the same general structure for each round but is parameterized by the round sub key K[*i*]. The key schedule algorithm is simple; the 128-bit key K is split into four 32-bit blocks K = (K[0], K[1], K[2], K[3]). The keys K[0] and K[1] are used in the odd rounds and the keys K[2] and K[3] are used in even rounds.

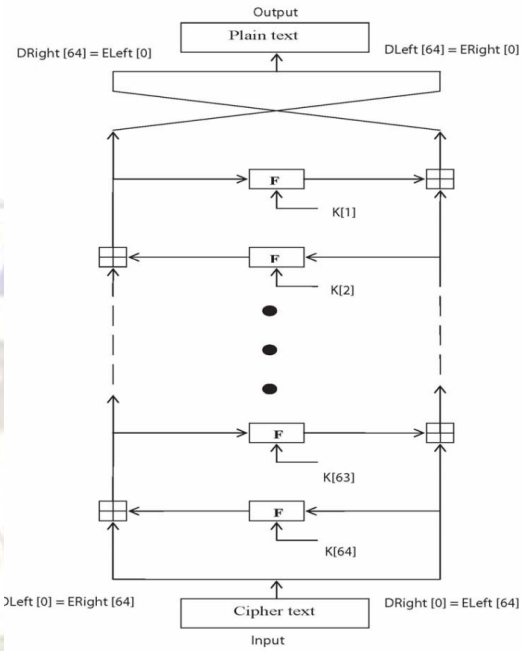

Fig.3: structure of the TEA decryption routine

Decryption is essentially the same as the encryption process; in the decode routine the cipher text is used as input to the algorithm, but the sub keys K[*i*] are used in the reverse order.

Fig.3 presents the structure of the TEA decryption routine. The intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. For example, if the output of the nth encryption round is

ELeft[*i*] || ERight[*i*] (ELeft[*i*] concatenated with ERight[*i*]).

Then the corresponding input to the (64-*i*)th decryption round is

DRight[*i*] || DLeft[*i*] (DRight[*i*] concatenated with DLeft[*i*]).

After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is ERight[64] || ELeft[64], the output of that round is the final cipher text C. Now this cipher text is used as the input to the decryption algorithm. The input to the first round is ERight[64] || ELeft[64], which is equal to the 32-bit swap of the output of the 64^{th} round of the encryption process.

Connection Manager: The Connection Manager deals with the architecture, which supports the system to identify the end users for the communication and establish the communication.

Connection and disconnection of the communication channel between the users for the access of file system and file transfer services. The Connection Manager receives the IP address to be connected and the file to be sent then establishes the connection and transfers the file.

DESIGN & IMPLEMENTATION

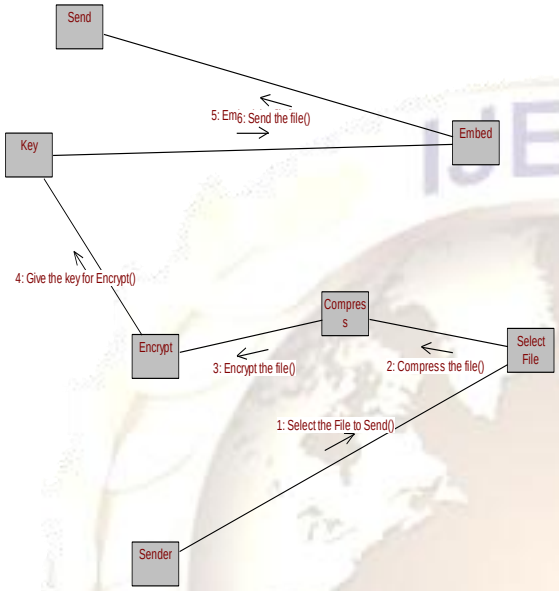

Fig.4: Interoperable Collaboration Diagram for Sender

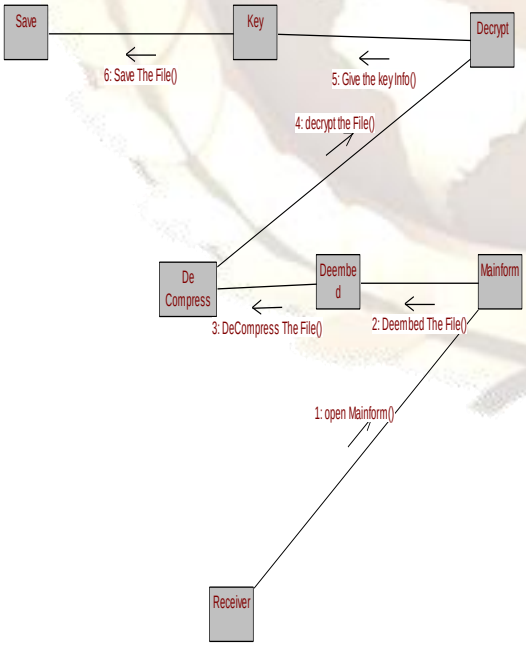

Fig.5: Interoperable Collaboration Diagram for Receiver

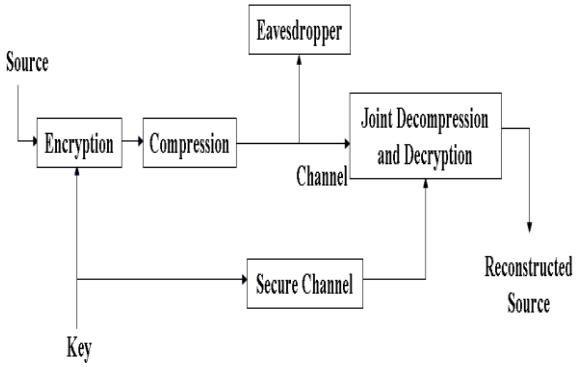

Fig.6: Interoperable Deployment Diagram

Below given code is used for digital encryption. Which is used by the sender to encrypt the code and transmit to the receiver.

```
Static String name;
public  DBS(int op,String x) {
String toBeSaved = "";
String theText;
int  choice=op;
name=x;
if(choice == 1) {
byte[] theFile = getFile(name);
String key = JOptionPane.showInputDialog("Enter your key (the longer the better):");
Encryption encryption = new Encryption(theFile,key);
encryption.encrypt();
saveFile(encryption.getFileBytes());
JOptionPane.showMessageDialog(null,"\nYour file has been encrypted and saved\n","message",JOptionPane.INFORMATION_MESSAGE);
}
else if(choice == 2)
{
byte[] theFile = getFile(name);
JPasswordField pf = new JPasswordField("Enter the key: ");
String key = JOptionPane.showInputDialog("Enter the key: ");
Encryption encryption = new Encryption(theFile,key);
encryption.decrypt();
saveFile(encryption.getFileBytes());
JOptionPane.showMessageDialog(null,"\nYour file has been decrypted and saved\n","message",JOptionPane.INFORMATION_MESSAGE);
}
else if(choice == 4)
{
System.out.println("GOOD DAY!");
System.exit(0);
}
else if(choice == 3)
```

```
{
String output="\n\nImportant info on key choice: \n\n"+
The longer the key, the better. This program\n" +
"Implements a key expansion algorithm that given\n" +
"an average length of user-entered key is almost\n" +
"analagous to the one-time pad encryption method\n\n" +
"For example: Use key length of 1: 128bit encryption\n\n"+
"Use key length of 2: 256bit encryption\n\n"+
"Use key length of 8: 1024bit encryption\n\n"+
"etc...\n\n\n";
JOptionPane.showMessageDialog(null,output,
"information",JOptionPane.ERROR_MESSAGE);
} }
```

RESULTS

The below are the obtained screen shots of the paper.

Fig.7: Main Screen to encrypt a file

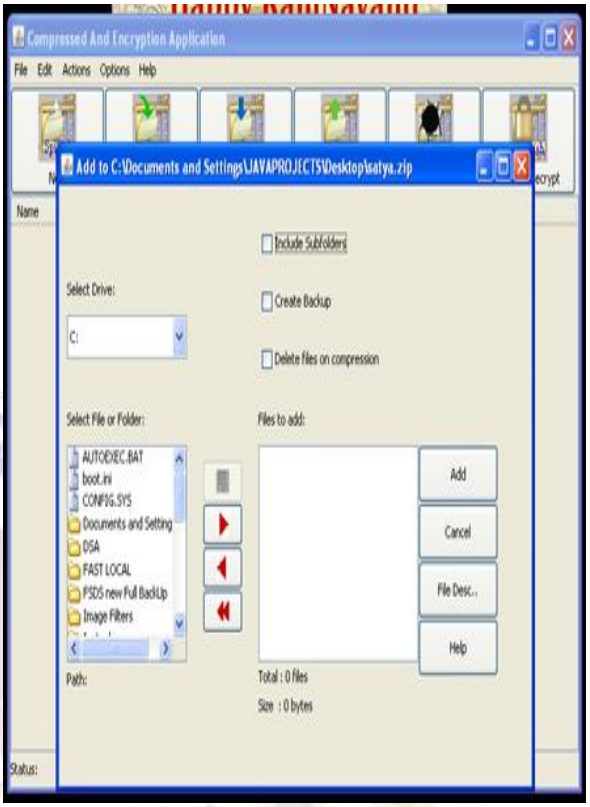

Fig.8 : Selecting a File to encrypt

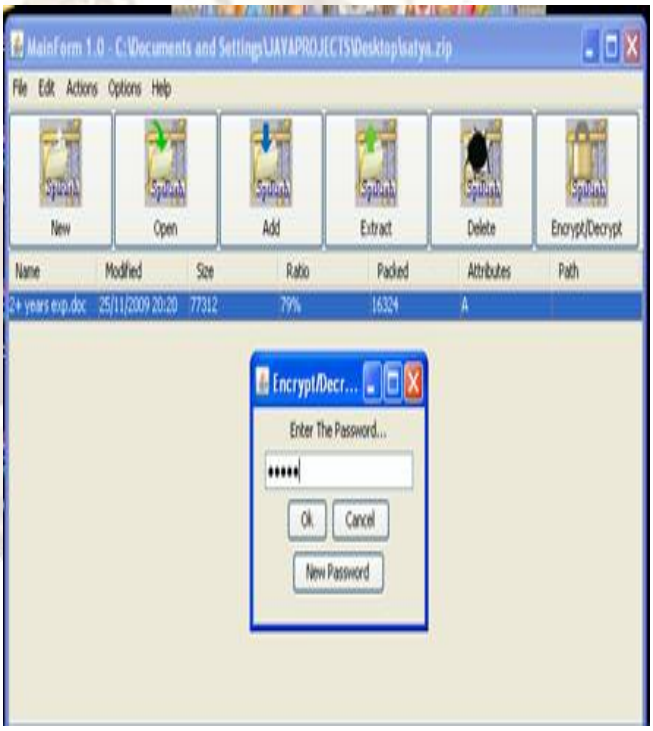

Fig.9: Process of Compressing & Encrypting the File using a password

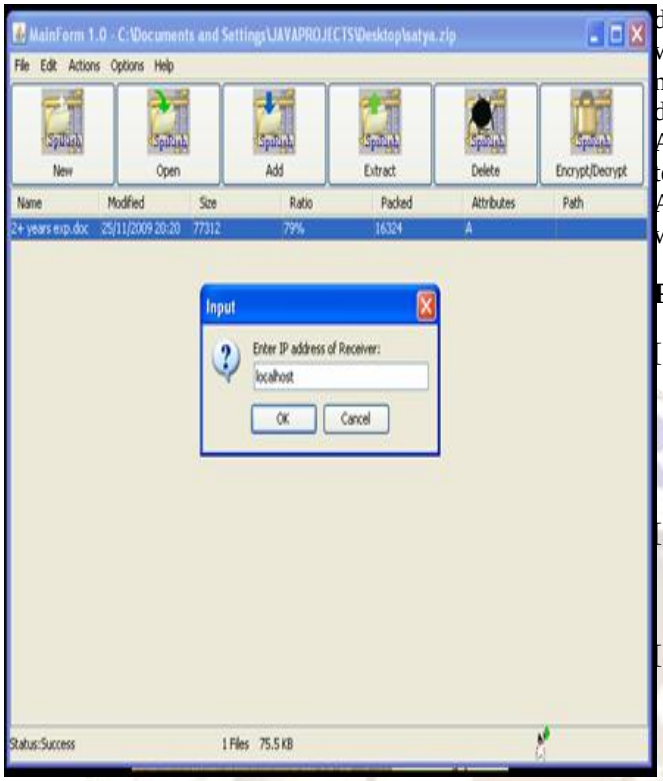

Fig.10: Giving the information of the receiver

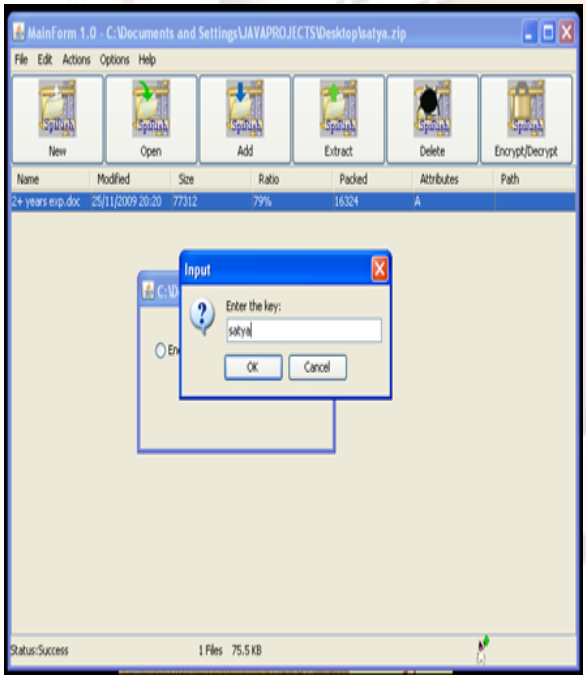

Fig.11: Receiver decrypting the file using the key

CONCLUSION

In this paper we address how Compression & decompression support security and cryptography to the data .For fast transmission of date and for security reasons we use compression & decompression techniques. The performs of compression decompression techniques for decrease the file size has been implemented which will increase the transmission speed. Here we managed to maintain compression and decompression technique based on the GZIP Algorithm and performed the cryptography technique based on the TINY Encryption Algorithm .So the security can be more provided with this technique during transmission of the data.

REFERENCES

[1] D. Coppersmith and M. Jakobsson, “Almost Optimal Hash Sequence Traversal”, Proceedings of the Fifth International Conference on inancial Cryptography, pp. 102-119, 2002.

[2] D. Dzung, M. Naedele, T.P. Hoff, M. Crevatin, “Security for Industrial Communication Systems”, Proceedings of the IEEE, vol. 93, no. 6, 2005

[3] J. Falco, J. Gilsinn, K. Stouffer, “IT Security for Industrial Control Systems: Requirements Specification and Performance Testing”, NDIA Homeland Security Symposium & Exhibition, 2004.

[4] M. Fischlin, “Fast Verification of Hash Chains”, The Cryptographers Track at the RSA Conference, pp. 339-352, 2004.

[5] J. Gilsinn, “Real-Time I/O Performance Metrics and Tests for Industrial Ethernet”, ISA Automation West, 2004.

[6] B. Groza, T.-L. Dragomir, "On the use of one-way chain based authentication in secure control systems", Second International Conference on Availability, Reliability and Security, pp. 1214-1221, IEEE Comp. Soc., 2007.

[7] N. Haller, C. Metz, P. Nesser, M. Straw, “A One-Time Password System”, RFC 2289, Bellcore, Kaman Sciences Corporation, Nesser and Nesser Consulting, 1998.

[8] O. C. Imer, S. Yuksel, T. Basar, “Optimal control of lti systems over unreliable communication links”, Automatica, (42), 2006.

[9] L. Lamport, “Password Authentication with Insecure Communication”, Communication of the ACM, 24, 770-772, 1981.

[10] C.J. Mitchell and L. Chen, “Comments on the S/KEY User Authentication Scheme”, ACM Operating Systems Review, pp. 12-16, 1996.