ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

Link Protection in DLNA Devices

(A secure cross-platform media sharing mechanism)

Pratheek Manjunath[1], Shruthi Sharath[2] Siddhant Gupta[3], Krishna Kishore[4]

Dept. of Telecommunication Engg, R.V College of Engineering, Bangalore, Karnataka, India [1]

Dept. of Telecommunication Engg, R.V College of Engineering, Bangalore, Karnataka, India [2]

Dept. of Telecommunication Engg, R.V College of Engineering, Bangalore, Karnataka, India [3]

Senior Network Architect, Sasken Communication Technologies, Bangalore, Karnataka, India [4]

ABSTRACT: The scenario of file transfers over various kinds of network is getting prevalent in today's time. Many protocols are used to transfer files over the network among various kinds of devices, few of them being DLNA, FTP, and e-mail. There is a need of developing an additional layer of security which prevents intrusion on the file transfer which can lead to theft or misuse. We have proposed a system to plug this loophole in security by introducing an additional layer of security at the application level.

The proposed system operates using blowfish algorithm to encrypt the data packets as they are streamed from one device to another. At the receiver, the incoming buffers are decrypted and written onto a file. The encrypting algorithm, though an entry-level one, generates ciphers which are considerably hard for amateur hackers to crack. The cryptographic dimension of this program can be augmented by using sophisticated algorithms on lines with AES or even incorporating digital signatures.

KEYWORDS: DLNA, Link Protection, Blowfish, Encryption, OpenSSL, Libcrypto, Application Security, Media Streaming

I. INTRODUCTION

In today's fast-paced world, people demand entertainment, productivity and services at their finger-tips while on-the-go. Smart phones have carved a niche simply because they cater to most of such needs at the comfort of our couch. But we're all aware that with convenience comes vulnerability and security risks.

There are many standards which are cross platform organization among the three electronics industry namely: Consumer Electronics, Personal Computers and Handheld devices. But protocols like DLNA standards provide integration for these 3 platforms. The vision of this alliance is for innovation which allows seamless environment for file sharing and growing new digital media as well as content services over both wired and wireless networks. All these standards have has published a set of common industry standards that enables the manufacturers to make products in the growing marketplace of network devices and allow these products to be interoperable over various open industry standards of digital media.

Wherever possible, these protocols refer to standards from established, open industry standards organizations and provide CE, PC and mobile device manufacturers with the information needed to build compelling, interoperable digital home platforms, devices and applications. [1]

With the advent of IP Enabled consumer electronics and mobile devices and the proliferation of digital media, the consumers are seeking ways to stream high definition videos.

The content being transferred is what is known as "Premium Content". This means that it was paid for, and probably licensed. Usually Premium Content carries with it restrictions about how it can be used, e.g. whether it can be copied or not. While the source and sink devices themselves may be able to enforce these restrictions, when the data is transferred over a network link it would be possible to circumvent them without Link Protection by copying or modifying the data directly from the network stream. For example, the Link Protection is implemented in DLNA using two existing Link Protection technologies for network devices – DTCP-IP and WMDRM-ND. These technologies provide a method to establish a secure channel between sources and sinks. Both technologies employ different techniques for authentication and content transfer, but they are fundamentally very similar. Using these methods,

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

Premium content transfer and media streaming over existing wireless networks do not provide the application level security. There is a need to develop a link protection technology. To tackle this problem, we propose a link protection application which can be installed on various enabled devices. The application will perform encryption & decryption along with file streaming over a network.

II. LITERATURE SURVEY

According to Kevin Arruda, in his paper titled "Link protection in DLNA", he explains link Protection as a blanket term that refers to the methods employed to contravene attempts by hackers to "steal" data from a data transfer link; whereas for DLNA, usually this means an Ethernet link in which media is transferred from a source to a rendering device. For an Unprotected Link, traffic is sent and received in "clear text", which means that anybody who can "see" the network traffic is free to copy and use the data on the link. Enabling link protection makes it possible to transfer media from the content source to a rendering device in a secure manner. [8]. As explained in the DLNA white paper on HD Video Streaming in Home Networking, Today, the home network is primarily used by consumers for a very short list of purposes. Consider the problems consumers have faced with printer sharing. While consumers could purchase a single printer to service an entire household, the complexity of initially configuring printers and later troubleshooting subsequent networking issues leads many users to connect multiple printers directly to individual PCs and laptops.

For digital audio, complexity of setup and lack of universal connectivity has made using independent docking stations a preferred alternative to streaming audio over the home network from a central storage location. If sharing printers and audio wasn't complicated enough for consumers, then sharing digital video, particularly HD video, can present an even more daunting task. Most video content has traditionally been stored on a PC or a digital STB. Not only must the PC or STB be turned on to access content, but they could be located in a different room than where consumers wish to sit, relax and watch. The reality is that sharing digital content including streaming HD video is practical using today's home networks. Reliable, quality streaming requires a high-bandwidth infrastructure, and many consumers already have an extensive home network infrastructure comprised of a variety of proven technologies like Wi-Fi, MoCA and Ethernet.

While these technologies enable powerful wired and/or wireless connectivity, connectivity involves so much more than just a physical infrastructure to route data. Across this infrastructure flows a myriad of content, link, transport, management, discovery, control and content protection protocols. In and of them, digital pipelines only move data and do little else to directly facilitate the sharing of content between connected devices

III. METHODOLOGY

Our methodology is to develop encryption/decryption protocols, using Libcrypto APIs, which is based on AES. The system can be developed using open source platforms like OpenSSL which is being used this system. The digital media is broken into buffers which are encrypted at the source and streamed over the network to the destination. The algorithm uses various elements of OpenSSL. Libcrypto, one of the primary libraries of OpenSSL provides both high level and low level interfaces for cryptographic algorithms. For most uses high level interface is used for performing cryptographic operations. The Envelope function provides the methodology to perform end to end cryptography, signing and verifying as well as generating digital signatures, hash functions and MAC Codes.

Public Symmetric Key Encryption

Symmetric key algorithm which encrypts and decrypts data using a single key is being used here. As shown in figure 1, the encryption algorithm takes the key and the plaintext as the parameters to produce the ciphertext. The ciphertext is then sent over a secure or non-secure medium, allowing only a recipient who has the original key to encrypt the message, which is done by passing the cipher text and the key to a decryption algorithm. Obviously, the key must be known only to the sender and the receiver. Figure 1shows symmetric key encryption model.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

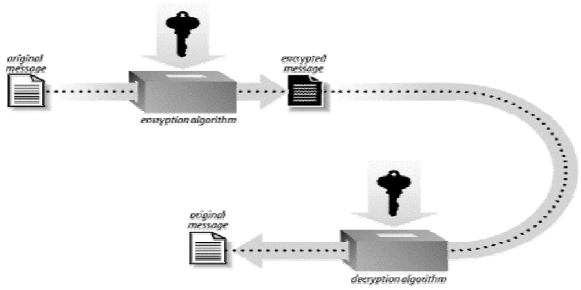

Fig .1 Symmetric Key Cryptography

The main disadvantage of this algorithm is that the key should be kept hidden throughout and should be known only to the recipient. It is very difficult to maintain the anonymity of the key on the same medium on which the file is being transferred. It leaves a possibility for any attacker to steal the key and gain access to the file. One solution to the key distribution problem is to use a cryptographic key exchange protocol. Open-SSL provides the Diffie-Hellman protocol for this purpose, which allows for key agreement without actually divulging the key on the network.

Public key cryptography can give another solution for the above said problem. In this method two keys are sent a public key which is freely distributed and a private key which is only known to the user.

Blowfish Algorithm

The cryptographic algorithm used for the encryption and decryption of data in the designed program is 'Blowfish'. Blowfish is a keyed, symmetric, cryptographic block cipher designed by Bruce Schneier. Primary features are 16-round Feistel network block cipher, a 64 bit block size and variable key length of up to 448 bits. The Blowfish algorithm is unpatented and is free for use by anyone is any application. The implementation consists of two parts: key-expansion and data encryption. The algorithm keeps two sub-key arrays: the 18-entry P-array and four 256-entry S-boxes. The S-boxes accept 8-bit input and produce 32-bit output. One entry of the P-array is used every round, and after the final round, each half of the data block is XORed with one of the two remaining unused P-entries. The diagram below shows Blowfish's F-function. The function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The outputs are added modulo 232 and XORed to produce the final 32-bit output. Blowfish is one of the fastest block ciphers in widespread use, except when changing keys. This is well suited for our entry-level program.

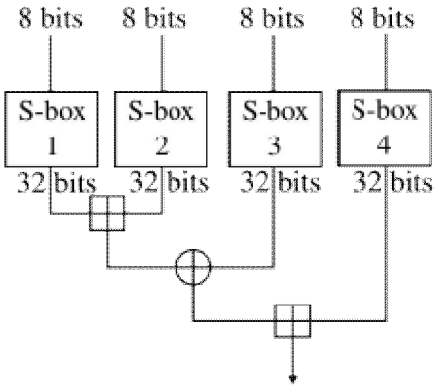

Fig .2 Fiestel Structure of Blowfish [7]

Cipher Block Chaining

Cipher Block Chaining provides the service of authenticity and security. [2] This mode describes how to apply the cipher for data repeatedly which is larger than the block size [3] [4] [5]. The initialization vector makes sure that a different ciphertext is produced when the same plaintext is given as input. [6]. IV is used to give random effect to the

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

encryption because the IV is produced from random numbers and hence can't be replicated. An initialization vector has different security requirements than a key, so the IV usually does not need to be secret. However, in most cases, it is important that an initialization vector is never reused under the same key.
In the proposed system, the cipher block chaining mode of encryption wherein a block of plain text, key and the IV are given as inputs and the cipher text is obtained as the output. This process of obtaining the cipher text is used for every block of plain text and every block of cipher text obtained is used in the process of obtaining the consecutive block of the cipher text.

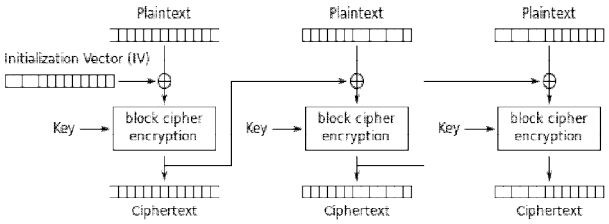

Fig .3 CBC Encryption

The same procedure is used for decryption as well. To get back the first block of plain text, IV, key (both transmitted along with the cipher text) and the first block of cipher text are used. Every consecutive block of plain text is got back using the same key and IV used at the transmission side (locally generated with the first key and IV that are transmitted) and the previous block of plain text retrieved.

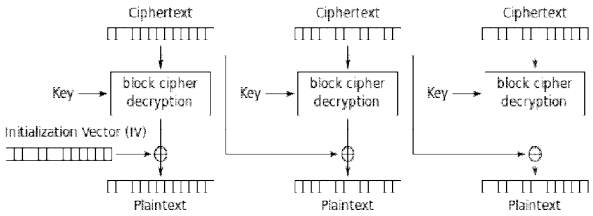

Fig .4 CBC Decryption

In CBC encryption, each cipher operation depends upon the previous operation except for the first one as it depends upon the original parameters. But on the decryption side the ciphers are immediately available for inverse operations. Even one-bit change in a plaintext or IV affects all following cipher text blocks. Decrypting with the incorrect IV causes the first block of plaintext to be corrupt but subsequent plaintext blocks will be correct.

Thus the complete encryption and decryption process consists of a key, an initialization vector (both generated using system noise), Blowfish algorithm as the encryption algorithm and cipher block chaining as the mode of encryption with buffers being used as the data structure for the entire process.

IV. IMPLEMENTATION

The media streaming between a Rendering device and a Server can be modelled using a Server-Client socket function. To simulate the working, we used socket functions on two PCs connected over a Local Area Network.
The entire process of establishing a connection, issuing a request, fetching the media file, writing data into buffer block, encrypting the buffers, streaming, decrypting the incoming buffer blocks, building the media file at the device and tearing down the connection happens as shown in the flow diagram (figure 2).

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

Below explained is the implementation of the technique used via the envelope (EVP) method under Open-SSL. The EVPs used for the encryption process are: EVP_Cipher_CTX_init, EVP_EncryptUpdate and EVP_EncryptFinal. Encryption begins with EVP_Cipher_CTX_init initializing the first buffer for the cipher text followed by EVP_EncryptUpdate till the last but one block of plain text. The last block of plain text uses EVP_EncryptFinal for encryption and terminates the encryption process. Similarly decryption is completed using EVP_DecryptInit, EVP_DecryptUpdate and EVP_DecryptFinal.

SERVER (PC 1)	CLIENT (PC 2)
Open Socket	
Bind to an address	
Listen to requests from client	
←	Send connection request to server
Establish connection	
Accept data request ←	Send Data request - file to be streamed
Open the file sought for	
Read from file and write to i/p buffer	
Encrypt contents of i/p buffer	
Send encryption parameters: key, i/p buffer length →	Receive encryption parameters: key, o/p buffer length
Send encrypted data →	Receive encrypted data and store in o/p buffer
	Decrypt o/p buffer using the parameters
	Write the decrypted data onto a file
Repeat encrypt and send until end of file is reached	Repeat receive and decrypt until last stream is received
→	Keep appending decrypted data into the target file
Teardown connection	
Close socket	

Fig .5 Flow Diagram for the implementation

The EVP functions provide a high level interface to Open-SSL cryptographic functions. The libcrypto library within Open-SSL provides functions for performing symmetric encryption and decryption operations across a wide range of algorithms and modes.

For the purposes of demonstration we have used a 128-bit key. This is stored as a character array in the program. We also generate a 64 bit initialization vector (IV). For our program we will use Cipher Block Chaining (CBC) mode as the mode of encryption. An initialization vector is a bit of random information that is used as an input in chained encryption algorithms. The IV provides the first bit of input for encrypting the 1st block of data, which then provides input for the 2nd block and so on. The bit left over at the end is discarded. The random bits are generated from the character special file which provides a good source for random numbers. We have also set up a buffer for the cipher text to be placed in. It is important to ensure that this buffer is sufficiently large for the expected cipher text or the program may crash. The cipher text may be longer than the plaintext (e.g. if padding is being used).

For encryption, the routine takes two parameters - the file descriptors of input file and the output file to which the encrypted data is to be saved. It is always a good idea to reset the buffers to zero before using them with data. This is especially needed if the buffers are being reused, which is the case here. Encryption consists of the following stages: Setting up a context, initializing the encryption operation, providing plaintext bytes to be encrypted and finalizing the encryption operation. During initialization, an EVP_CIPHER object message is provided. The decryption routine basically follows the same steps as the encryption routine. This is very similar to encryption and consists of the following stages: Setting up a context, initializing the decryption operation, providing cipher text bytes to be decrypted and finalizing the decryption operation. Again through the parameters we will receive the cipher text to be decrypted, the length of the cipher text, the key and the IV. We will also receive a buffer to place the decrypted text into, and return the length of the plaintext we have found. Every time the length of the cipher text is passed. This is required as

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

we cannot use functions such as "strlen" on this data because it is binary. Similarly, even if the plaintext is ASCII text, Open-SSL does not know that. In spite of the name, plaintext could be binary data, and therefore no NULL terminator will be put on the end (unless the NULL is also encrypted).
The process of encryption and decryption use the following EVPs.

Initially, the process of encryption begins with EVP_CIPHER_CTX_init () initializing the cipher content "ctx". EVP_EncryptUpdate (&ctx, outbuf, &olen, inbuff, n) is used to Encrypts *inl* bytes from *buffer in* and write the encrypted version to *out*. It can be called multiple times to encrypt successive blocks of data. The amount of data written depends on the block alignment of the encrypted data. Amount of data written may be anything from zero bytes to *(inl +cipher_block_size – 1)*.

If padding is enabled (default) then EVP_EncryptFinal (&ctx, outbuf+olen, &tlen) encrypts the final data, that is, any data that remains in the final partial (or full) block. The encrypted final data is then written to *out* which should have sufficient space for one block of cipher and *ctx* is automatically cleaned up after the call. If padding is disabled then EVP_EncryptFinal (&ctx, outbuf+olen, &tlen) will not encrypt any more data and it will return an error if any data remains in a partial block, that is if the total data length is not a multiple of the block size.
EVP_DecryptInit (&ctx, EVP_bf_cbc (), key, iv), EVP_DecryptUpdate (&ctx, outbuf, &olen, inbuff, n) and EVP_DecryptFinal (&ctx, outbuf+olen, &tlen) are the corresponding decryption operations.
EVP_DecryptFinal () will return an error code if padding is enabled and the final block is not correctly formatted. The parameters and restrictions are identical to the encryption operations.
EVP_CipherInit (), EVP_CipherUpdate () and EVP_CipherFinal () are functions that can be used for decryption or encryption.

The operation performed depends on the value of the *enc* parameter in the function. It should be set to 1 for encryption, 0 for decryption and -1 to leave the value unchanged.
EVP_CIPHER_CTX_cleanup () is used to clear all information from cipher context and free any allocated memory associated with it. It should be called after all the operations using a cipher are completed so that sensitive information does not remain in memory.

V. RESULTS OBTAINED

Results obtain from the designed system were as follows.

A. Encryption and Decryption

The encryption and decryption operations are performed on complete files. This stage doesn't incorporate streaming.
Source file-The figure shows the source file which is an image. File name is sendi.jpg and its size is 1Mb.

Fig .6 Source file- sendi.jpg

Encryption Terminal-The figure shown is of the encryption terminal. The inputs to the program should contain the source file path and a destination path to save the encrypted file.
Source file: sendi Destination file: enci
The output displays the 128bit key generated and the Initialization Vector

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

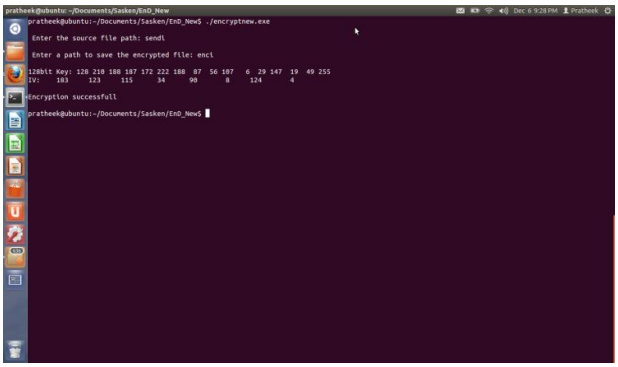

Fig .7 Encryption Terminal

Encrypted File

The encrypted file 'enci' created by the encryption program is not of any recognizable format. If opened in the text editor, it looks like this:

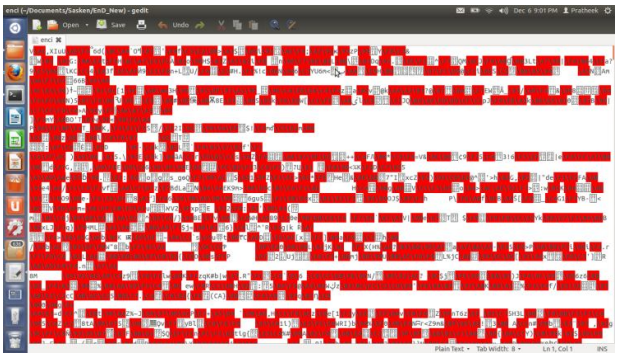

Fig .8 Encrypted File

Decryption Terminal

The figure shown is of the decryption terminal. The inputs to the program should contain the encrypted file path and a destination path to save the decrypted file.

Source file: enci Destination file: recvi

The output displays the 128bit key generated and the Initialization Vector. It can be verified that the Key and IV are the same as those generated by the encryption program.

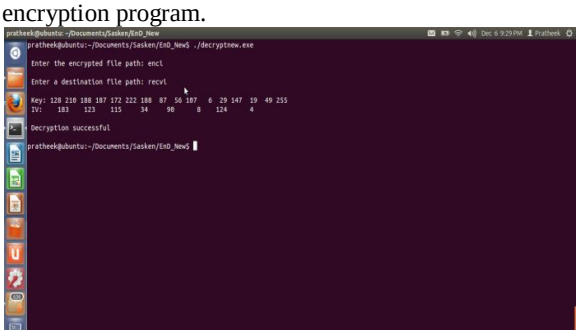

Fig .9 Decryption Terminal

Decrypted file

The decrypted file is of .jpg format, same as the source file. It can be verified that there isn't any change in the resolution or colour properties.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

Fig .10 Decrypted file-recv.jpg

B. Encryption and Decryption with Streaming from server to client

This stage incorporates streaming for the encryption and decryption programs. Both client and server terminals must be open in order for a connection to be established.

Text File-This is the source file, a text file of size 0.8 Mb. This is to be encrypted and streamed.

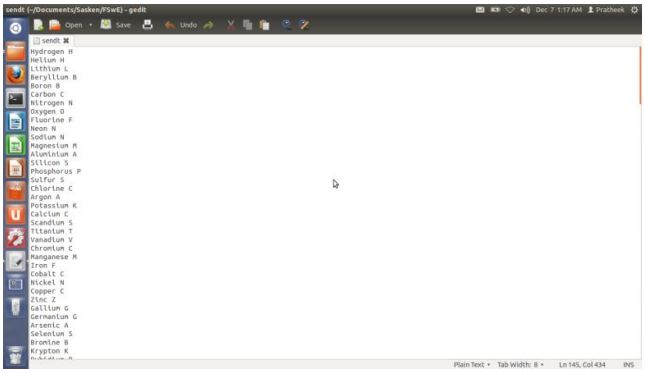

Fig .11 Text file which will be streamed

Encryption Terminal and Buffering Terminal

First, the client sends a connection request to the server. The listening server accepts the request and binds to the client address. After socket is opened, the client sends the designated text file to the server. Buffers of size 1024 bytes are encrypted and streamed until end of file is reached.

The input to this terminal must specify the source file path 'sendt'. The terminal prompts for the file path as soon as the socket has been opened.

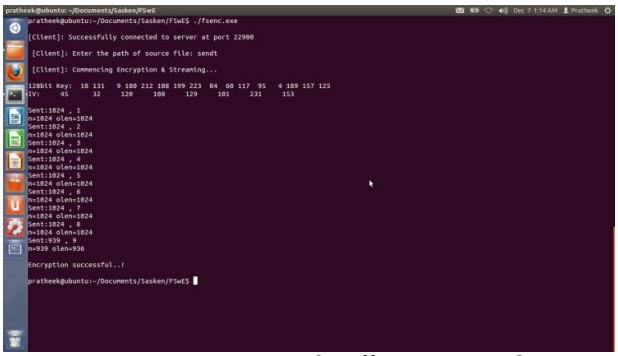

Fig .12 Encryption and Buffering Terminal

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

Receiving and decryption terminal

The server terminal must be running before encryption commences. After it is connected to the client, the program prompts for a destination file name and path. Upon entering this, reception and decryption of the buffers begin. The resulting data is appended to build the destination file.

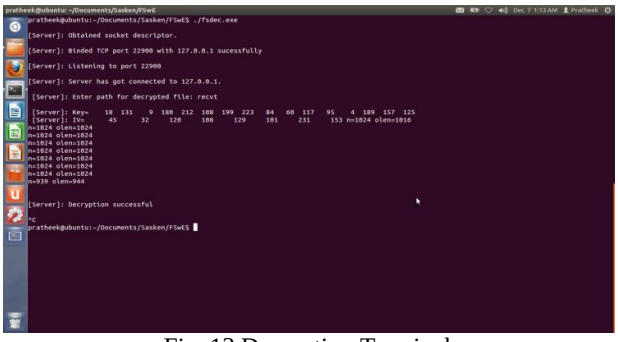

Fig .13 Decryption Terminal

Decrypted file

The received text file is built as and when buffers arrive. The data is written onto the text file 'recvt'. It may be observed that the source file 'sendt' and received file 'recvt' have identical contents and are of same size.

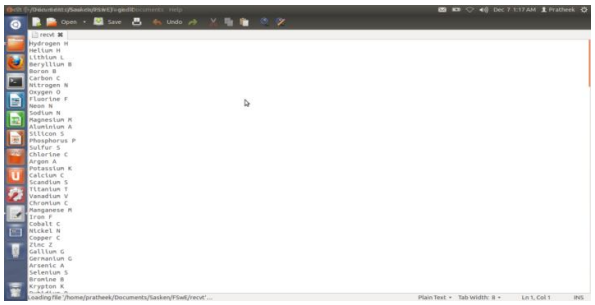

Fig .14 Decrypted File

1. Buffer Size versus Time taken to stream over LAN

Sl No	Test Cases (size=71.885MB)	Time taken for complete file transfer over LAN
1	512	5.340 seconds
2	1024	3.240 seconds
3	2018	2.810 seconds
4	4096	2.250 seconds
5	8192	2.140 seconds
6	16384	1.780 seconds

The above table shows the time taken for the complete file transfer for various buffer sizes. It can be seen that as the buffer size increases time taken for the file to be transferred decreases.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

2. File size versus Time taken to encrypt & decrypt

Test Cases (Buffer=1024 bytes)	Encryption	Decryption
1. Text file-660.496 KB	0.0200 s	0.0100 s
2. Image-1.132MB	0.0300 s	0.0200 s
3. Music clip-3.311 MB	0.0600 s	0.0500 s
4. Video clip-71.885 MB	1.3800 s	1.1800 s

The above table shows the time taken to encrypt and decrypt for different kinds of file.

3. File size versus transfer duration over a LAN

Test Cases (Buffer=1024 bytes)	Encryption, Streaming and Decryption
1. Text File-660.496 KB	1.030 seconds
2. Image-1.132MB	1.060 seconds
3. Music clip-3.311 MB	1.140 seconds
4. Video clip-71.885 MB	3.240 seconds

The above table shows the total time taken for the overall process to complete for various kinds of media files

VI. APPLICATIONS

The main application of this algorithm is to encrypt the data at the sender's side, transmit it securely across a link and finally decrypt it at the receiver's end. It can be used in different ways as follows: It can be used to encrypt media (text, image, audio or video) files as a whole, transmit them securely and decrypt them at the receiver's end. The source file's name and the encrypted file's name need to me mentioned at the time of encryption. Using the algorithm, the key and IV are generated. At the time of decryption, encrypted file path and a destination path to save the decrypted file need to be mentioned. It can be seen that the key and IV are the same as those generated at the encryption end.

Another application of this algorithm is streaming can be incorporated for both the encryption and decryption programs. Both client and server terminals must be open in order for a connection to be established. First, the client sends a connection request to the server. The listening server accepts the request and binds to the client address. After socket is opened, the client sends the designated text file to the server. Buffers of size 1024 bytes are encrypted and streamed until end of file is reached. The server terminal must be running before encryption commences. After it is connected to the client, the program prompts for a destination file name and path. Upon entering this, reception and decryption of the buffers begin. The resulting data is appended to build the destination file.

This can be used in real time applications like chatting for people in far of locations but connected across a network, to share complete files of any kind (multimedia) securely (secured wireless streaming) through emails and sharing the key separately through another email or SMS.

VII. LIMITATIONS

A few limitations of this system are, for the process of encryption, symmetric key algorithm is being used. A major disadvantage of this kind of algorithm is that the key that is used for decryption at the receiver side has to be the same as the key used for encryption at the sender side. Thus the key has to be transmitted to the receiver side through the same or a different network securely. Any compromise in security for this transmission leads to complete failure of the system. Another reason why this solution needs some re-thinking is the issue with respect to interoperability. In case of DLNA, the solution works only between devices with proprietary solutions hence achieving interoperability with become difficult.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2014

One more shortcoming of this system is that the algorithm used is of a very basic level and thus algorithms that provide higher levels of security can be used to enhance the performance of this system. A few alternatives are Advanced Encryption Standard (AES), Twofish Algorithm, Threefish Algorithm, Salsa20, Tiny Encryption Algorithm (TEA) and CMAC and MacGuffin. Twofish algorithm is known for its wide range of speed and cost tradeoffs; it offers a unique combination of conservative design, speed and flexibility. Three fish is a symmetric-key block cipher designed as part of the Skein hash function. Three fish uses no S-boxes or other table lookups in order to avoid cache timing attacks.
Salsa20 performs 20 rounds of mixing on its input. However, reduced round variants Salsa20/8 and Salsa20/12 using 8 and 12 rounds respectively have also been introduced. These variants were introduced to complement the original Salsa20, not to replace it, and perform even better than Salsa20, though with a correspondingly lower security margin.
TEA algorithm will easily translate into assembly. The hardware implementation is not difficult, and is of the same order of complexity as DES, taking into account the double length key.
CMAC is a block cipher based message authentication code algorithm and MacGuffin block cipher is used for this implementation.

Possible future work includes implementing CMAC with a different block cipher (AES for example). A user defined tag length for CMAC could be implemented in a future version. This would allow the user to manage the performance and security trade-off.

REFERENCES

We express our gratitude towards our guide, Professor P. Nagaraju of the Dept. of Telecommunication Engineering, RVCE, for his innovative ideas and technical advice.

[1] "DLNA Networked device interoperability guidelines", Copyright © 2013 Microsoft Corporation, Release: Monday, July 22, 2013
[2] "Proposed modes", NIST Computer Security Division's (CSD) Security Technology Group (STG) (2013). Cryptographic Toolkit. NIST. Retrieved April 14, 2013.
[3] ISO JTC 1/SC 27 (2006). ISO/IEC 10116:2006. ISO Standards catalogue.
[4] Kuo-Tsang Huang, Jung-Hui Chiu, and Sung-Shiou Shen, "A Novel Structure with Dynamic Operation Mode for Symmetric-Key Block Ciphers", International Journal of Network Security & Its Applications (IJNSA) 5 (1): 19, January 2013.
[5] "Current modes", NIST Computer Security Division's (CSD) Security Technology Group (STG) (2013). Cryptographic Toolkit. NIST. Retrieved April 12, 2013.
[6] "Stream Cipher Reuse: A Graphic Example". Cryptosmith LLC. Retrieved 27 March 2013.
[7] https://www.schneier.com/blowfish.html
[8] Open-SSL Homepage – http://www.Open-SSL.org
[9] Open-SSL lab session by Marco Tiloca - marco.tiloca@iet.unipi.it
[10] Alfred J. Menezes, Paul C. Van Oorschot and Scott A. Vanstone , "Information technology -- Security techniques -- Modes of operation for an n-bit block cipher", Handbook of Applied Cryptography. CRC Press. pp. 228–233. ISBN 0-8493-8523-7, 1996
[11] Ferguson N., Schneider, B. and Kohno, T, "Cryptography Engineering: Design Principles and Practical Applications", Indianapolis: Wiley Publishing, Inc., ISBN 978-0-470-47424-2, Vol 64, pp. 63, 2010.
[12] Kevin Arruda, "Link protection in DLNA", © Copyright 2008 Lamprey Networks Inc.