The Impact of Frame Rate on Securing Real Time Transmission of Video over IP Networks

NOUR EL DEEN M. KHALIFA, HESHAM N. ELMANHY

Information technology Department
Faculty of Computers and Information - Cairo University
5 Dr. Ahmed Zuwail Street, Orman
Giza, Postal code 12613. EGYPT

nourmahmoud@gmail.com
ehesham@mailer.eun.eg

Abstract - The nature of playing video streams over a network in a real time requires that the transmitted frames are sent in a bounded delay. Also, video frames need to be displayed at a certain rate; therefore, sending and receiving encrypted packets must be sent in a certain amount of time in order to utilize the admissible delay. There are two main components for securing multimedia transmissions. Key management protocols such as Transport Layer Security (TLS) protocol is used as cryptographic protocol that provides security and data integrity for communications over IP Networks. Advanced Encryption Standard (AES) is used as a symmetric block cipher, that can encrypt (encipher) and decrypt (decipher) video frames. The Real-time Transport Protocol (RTP) protocol is used as a standardized packet format for delivering video over the IP Networks. This makes secure video encryption feasible for real-time applications without any extra dedicated hard-ware. In this paper; we studied the impact of frame rate on a secured real time transmission of video over IP networks. Through this paper, we will try to find the most suitable frame rate in order to achieve better data rate and fewer frame and packet loss. This is done via java as a programming language for implementation.

Keywords: Transport Layer Security (TLS); Real-time Transport Protocol (RTP); Advanced Encryption Standard (AES); Frame Rate.

1. INTRODUCTION

The advent of networked multimedia systems will make continuous media streams, such as real time audio and video, increasingly pervasive in future computing and communications environments. It is thus important to secure real time video streaming over IP networks from potential threats such as hackers, eavesdroppers, etc [1]. This work was originally done at section 5, 6. Consequently, we study the impact of frame rate on securing video streams. Also, we study the performance of encryption and decryption algorithms such as AES for real time video streams. We present our results in diagrams, which shows the encryption performance between different frame rates and what we have to pay to grantee a secure transmission of video over IP networks. In short, the foremost goal of this research is to cover these points:

• Determining how key management protocols such as Transport Layer Security (TLS) protocol can be implemented and what are the advantages and disadvantages of this protocol.
• Determining how encryption and decryption can be implemented for the real time application and study the impacts on received video frames.
• Determining what is the best frame rate that achieves better data rate and fewer frame and packet loss.
• Comparing our results with the previous results in video encryption.
• Computing performance and overheads of multimedia security.

2. REAL-TIME TRANSPORT PROTOCOL (RTP)

The Real-time Transport Protocol (RTP) is a standardized packet format designed to deliver media contents (mostly video and audio, but also images) over an IP network [2]. It was first defined in 1996 as RFC 1889, which was replaced by RFC 3550 in 2003. It was originally thought as an add-on for the UDP protocol in media environments, although it can be also delivered over a TCP layer. It has no defined port to be delivered, so it is usually delivered in the wide area of non defined ports. However, it is said in the standard that RTP must use an even port and the Real-time Transport Control Protocol (RTCP) use the next odd port [2].

An RTP packet is divided into a 12-bytes header and a payload (see Figure 1). The payload would carry the data of the packet and in the header the following information is provided [2]:

- **RTP Version [V]** (2 bits): indicates the version of the protocol. The current version is 2.
- **Padding [P]** (1 bit): indicates if there is extra information at the end of the packet, in order to exactly fit on an 8 bit alignment.
- **Extension [X]** (1 bit): indicates if any extension of the protocol is used in the packet.
- **Contributors [NCSRC]** (2 bits): indicates the number of contributors to that session. It is used to read the CSRC optional headers.
- **Marker bit [N]** (1 bit): it is used to inform the application that something special is in that packet. Usually is always off in audio streams and 1 in video packets containing the last information of a frame.
3. MODIFIED REAL-TIME TRANSPORT PROTOCOL (RTP)

In order to implement a secured RTP, minor changes have been done on the RTP packet format. Replace contributing source identifier (CSRC) (16 bits) field with encryption algorithm type (ET) (16 bits) field.

This change is not considered as an overhead, as the contributing source identifier (CSRC) field is an optional header. Note that the RTP payload holds encrypted data; this will be discussed in section 6.

4. MULTIMEDIA FORMAT (M-JPEG)

In multimedia, Motion JPEG (M-JPEG) is an informal name for multimedia formats where each video frame or interlaced field of a digital video sequence is separately compressed as a JPEG image. M-JPEG is considered one of the best multimedia formats for our implementation as it allows us to handle frames of video, also commonly used by IP based video cameras via RTP streaming.

The main advantages are low processor overhead and video stream is extremely easy for editing and encoding. The edited M-JPEG stream into final format (such as MPEG-1, MPEG-2) gives normally the best possible video quality. The main disadvantages relatively high bit rate for the delivered quality, requiring more storage space than more modern formats (such as JPEG 2000) for a given image quality.

5. SECURITY

Security is the condition of being protected against danger, loss, and criminals. Studies have shown that users may be more sensitive to privacy concern with audio and video communication than have been with more traditional forms of network communications.

The Java platform's security and encryption features have grown tremendously over the last few years. The Java Development Kit (JDK 1.4) release now comes bundled with many security-related packages, including the Java Cryptography Extension (JCE), the Java Secure Socket Extension (JSSE), and the Java Authentication and Authorization Service (JAAS). All of these components are pieces of the Java Cryptography Architecture (JCA) [4], [5], as illustrated in the Figure 2.

5.1. JAVA SECURE SOCKET EXTENSION (JSSE)

JSSE provides secure socket communication for the Java 2 platform. More precisely, it implements Secure Socket Layer (SSL) and Transport Layer Security (TLS), two standardized protocols for implementing secure communications over the Internet. Both SSL and TLS rely on public-key cryptography [6].

In our implementation, we used Transport Layer Security (TLS) protocol to establish a secure connection between server and client.

TLS protocol location in the Internet protocol stack between application layer and transport layer as illustrated in Figure 3.

TLS protocol uses public-key cryptography to exchange a set of shared keys, and then uses standard shared-key encryption to exchange data. The shared keys are used both for encrypting the data (making it unreadable by others) and for authenticating the data (ensuring that it hasn’t come from an impostor) [7].
Before data can be sent across a TLS connection, the two ends must negotiate and exchange key information. This is called the handshake protocol. We won't go into much detail about the handshake protocol here, because it isn't necessary for our purposes. For our purposes, you need to know that the handshake involves the following steps:

1) The server sends its certificate to the client and the client verifies the server certificate.
2) The client sends its certificate to the server and the server verifies the client certificate.
3) The client encrypts password information with the server's public key and sends it to the server. This password information is used by each end of the connection to generate identical secret keys, which will then be used to transmit data.

Client authentication (Step 2) is optional, the server can request that the client provide its certificate, but it is not required to make such a request. We will be using client authentication in our implementation.

5.1.1. KEY MANAGEMENT IN JSSE

Key generation and manipulation is performed with the keytool program, which is included with the JSSE packages in JDK 1.4. keytool can be used for a variety of purposes. Here, we'll be using it to create public/private key pairs, and to extract the public key certificates from these pairs and place them in their own files. The following keytool command is used to generate a new public/private key pair:

```
keytool -genkey -keystore [filename]
```

When you run this command, you will be asked a series of questions. These questions about (your name, organization, and the like). The information you provide will be used to create a self-signed certificate that associates the information with a public key and testifies to the authenticity of the association. You will also be asked to enter passwords for the keystore and, optionally, passwords for the key pair you are creating.

The first step in the preparation of our implementation is to generate a public/private key pair for the client and another pair for the server. The following command will generate the file `client.private`, which is the key pair for the client:

```
keytool -genkey -alias clientprivate -keystore client.private -storetype JKS -keyalg rsa -dname "CN=Your Name, OU=Your Organizational Unit, O=Your Organization, L=Your City, S=Your State, C=Your Country" -storepass clientpw -keypass clientpw
```

And here is the command to generate the file `server.private`, which is the key pair for the server:

```
keytool -genkey -alias serverprivate -keystore server.private -storetype JKS -keyalg rsa -dname "CN=Your Name, OU=Your Organizational Unit, O=Your Organization, L=Your City, S=Your State, C=Your Country" -storepass serverpw -keypass serverpw
```

The private client and server key files need to be installed in specific places. The `client.private` file is installed on the client side; the `server.private` file is installed on the server side.

Key files should be installed on the client side:
- `client.private`
- `server.public`

Key files should be installed on the server side:
- `server.private`
- `client.public`

After creating key files, note that this process is done once the program executed for first time. We initiate a secure socket connection to a remote server; we must carry out the following steps:

1) Create a `SecureRandom`, a source of secure random numbers. Secure random numbers are numbers that are random enough that they will not make the encryption vulnerable to attack.
2) Create a `KeyStore` object containing the server's public key. This is read from `server.public`.
3) Create a `KeyStore` object containing the client's public/private key pair, including its public key certificate. This is read from `client.private`.
4) Create a `TrustManagerFactory` from the remote server's `KeyStore`. This is used to authenticate the remote server.
5) Create a `KeyManagerFactory` from the client's `KeyStore`. This is used for encrypting and decrypting data.
6) Create an `SSLContext` object, using the `KeyManagerFactory`, the `TrustManagerFactory`, and the `SecureRandom`.
7) Use the `SSLContext` to create an `SSLSocketFactory`.
8) Use the `SSLSocketFactory` to create an `SSLSocket`, which acts just like a regular Socket, except that it is secure.

To listen for incoming connections, we must carry out a similar set of steps:

1) Create a `SecureRandom`, a source of secure random numbers.
2) Create a `KeyStore` object containing the remote client's public key. This is read from `client.public`.
3) Create a `KeyStore` object containing the server's public/private key pair, including its public key certificate. This is read from `server.private`.
4) Create a `TrustManagerFactory` from the remote client's `KeyStore`. This is used to authenticate the remote client.
5) Create a `KeyManagerFactory` from the server's `KeyStore`. This is used for encrypting and decrypting data.
6) Create an `SSLContext` object, using the `KeyManagerFactory`, the `TrustManagerFactory`, and the `SecureRandom`.

59
notice some advantages and disadvantages. The major advantage is the secured connection will prevent some famous attacks such as Downgrade Attack, Truncation Attack, padding-oracle attack, timing attack, man-in-the-middle (MITM) attack [7]. On the other hand, the major disadvantage is the long setup time for establishing a secured connection, this will discuss later in experiments and results section.

5.2. JAVA CRYPTOGRAPHY EXTENSION (JCE).

The process of converting plaintext to cipher text is called enciphering or encryption; restoring plaintext from cipher text is deciphering or decryption. Both the encryption and decryption algorithms take a key (K) and plain text/cipher text as input. In the case of images, plaintext is a set of pixel values arranged in an orderly manner. Encrypting images/videos constitutes reordering these pixel values so that they convey no visual information about the original image.

In cryptography, a block cipher is a symmetric key cipher which operates on fixed-length groups of bits, termed blocks, with an unvarying transformation. When encrypting, a block cipher might take an n-bit block of plaintext as input and output a corresponding n-bit block of cipher text. The exact transformation is controlled using a second input — the secret key. Decryption is similar, takes an n-bit block of cipher text together with the secret key and outputs the original n-bit block of plaintext. Examples of block ciphers are RC5, AES, DES, Blowfish, etc [6]. We use block ciphers such as AES in this work, Figure 4 illustrates the process of encryption/decryption for block cipher algorithm.

A block cipher operates in different modes. The predominant modes are Electronic Code Book (ECB) and Cipher-Block Chaining (CBC). In ECB mode, the plaintext is divided into blocks and each block is encrypted separately. In the CBC mode, each block of plaintext is XORed with the previous cipher text block before being encrypted. This way, each cipher text block is dependent on all plaintext blocks processed up to that point. In this mode, changes in the plaintext propagate forever in the cipher text and encryption cannot be parallelized. Also, decryption cannot be parallelized [6]. In our Implementation (ECB) is used as it is faster than (CBC), this noticed from experiments.

The Java Cryptography Extension (JCE) provides a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. The package also supports secure streams and sealed objects [6].

5.2.1. ADVANCED ENCRYPTION STANDARD (AES).

In our implementation, we choose AES as default algorithm this is due to the following:
1) AES can be implemented to run at speeds unusually fast for a block cipher on a Pentium (Pro).
2) The round transformation is parallel by design, an important advantage in future processors and dedicated hardware.
3) As the cipher does not make use of arithmetic operations, it has no bias towards big or little endian processor architectures.
4) The tight cipher design does not leave enough room to hide a trapdoor.
5) The design allows the specification of variants with the block length and key length both ranging from 128 to 256 bits in steps of 32 bits.
6) Although the number of rounds of AES is fixed in the specification, it can be modified as a parameter in case of security problems.

Disadvantage of AES through implementation is the process of Padding, extra bytes added into frame to become balance for key size in order to complete process of encryption.

In our implementation AES encryption/decryption done on the level of video frames, and gives very good results in time spent for encryption and decryption and this will be discussed in details in section 7.

6. OUR IMPLEMENTATION

We would like to mention that through our literature review for previous work our paper is the first one that implements a security model using java programming language, and the first paper that studies the impact of frame rate on securing video over IP networks.

Figure 5 illustrates the block diagram for the proposed model through the following steps:
1) A Secured Connection established using TLS protocol, discussed in section 5.1.1.
2) Server generates a random key (Size of 16 byte) for encryption/decryption and sends it to client through the secured connection, which is created in step 1.
3) Server starts to read video stream and use a framing function (extract frames from video stream)
4) The extracted frame is passed to an encryption function, discussed in section 5.2.
5) RTP protocol initiates its header values, illustrated in Figure 1, and builds RTP packets for encrypted frame, then sends them to client.
6) Client starts to receive encrypted packets, and builds an encrypted frame using sequence number filed.
7) Using a decryption function, the encrypted frame returns to the original one.
8) Client starts to display frames in a certain frame rate determined by server.

7. EXPERIMENTAL RESULTS

In order to evaluate the performance of the proposed model and determining the best frame rate to be used in securing video over IP networks, we implemented the model on a 2 GHz dual-core processor with a RAM of 3 GB as a server, a 2 GHz processor with a RAM of 1 GB as a client, and 10/100 megabyte network.

The model is coded in Java using the JCA library, which supports JCE and JSSE, and Java Media Framework (JMF) library that supports RTP protocol and MJPEG codec. The video data used for analysis has different motion characteristics, varying data size and screen resolution. The sample test video sequences include videos like web cameras videos, parts from famous movies... etc.

The time spent for a secured connection setup in our implementation is around 2.31 seconds per connection. Since this is done infrequently (only when key is renewed) and is an off-line process, it does not affect the encryption time of the video. Figure 6 illustrates time spent to establish a normal connection with RTP protocol and a secured connection with a modified RTP (Secured RTP).

Apart from providing security (which is explained in Section 5.2.), an encryption algorithm for real-time multimedia data should also have two characteristics. Firstly, the encryption time should not be high, as it delays the transmission. Secondly, the frame size of the video should not increase so much, making it difficult to transmit over a network. The AES algorithm possesses both the characteristics with much less overhead on the frame size, discussion of these characteristics follows in details.

The encryption time spent by our implementation is quite low and suited for real-time applications, as it takes around 1.12ms per frame for encryption and decryption process in both server and client PC’s. Results will be compared to other algorithms later in section 7.1. Figure 7 illustrates the spent time for encryption/decryption process for secured RTP and spent time for frame in normal RTP.
After number of experiments, we notice that the frame rate has an impact on the frame and packet loss. Figure 8 illustrates that the best frame rate is 22 that has no frame loss.

When frame rate is less than or equal 22, the RTP protocol has the same results with SRTP in the number of frames received by the client.

Also when frame rate is less than or equal 22, the RTP protocol has near results with SRTP in the number of packets received by client, noted that the RTP packet length is equal to SRTP packet length as illustrated in section 3. SRTP packet length is 128 bytes (120 bytes for encrypted payload and 8 bytes for header), as illustrated in section 2. Figure 9 illustrates what we have mentioned before for packet rate, and noted that when frame rate is less than or equal 22, the packet rate of SRTP is little more high than packet rate of RTP. This is due to padding process that was discussed in section 5.2.1.

Data rate also is affected by padding, it is clearly shown in Figure 10, when frame rate is less than or equal 22, the data rate for SRTP is little more high than data rate for RTP; this is due to padding process also.

7.1 COMPARING ENCRYPTION SPENT TIME

As mentioned before, this paper is the first that implements AES over video frames using java, so it was very difficult to find any related work, but we find a paper released in December 2008 that implements a new technique for encryption but using C programming language [8]. We compare our results with this paper as illustrated in Figure 11, it can be observed that the average encryption time per frame taken by AES is around 1.12ms, which is less than that of other techniques like RC5 (1.89ms) [9], SRMT [10] (1.86ms) and XOR followed by scrambling (1.26ms) [11],Paper algorithm (1.38ms) [8].

7.2 THE IMPACT OF ENCRYPTED VIDEO SIZE

The Encrypted video size has no impact on video transmission over network and can be ignored as a key parameter, this is due to: the encrypted video size is only 0.099% bigger than the original video size, this value is mathematically calculated and can be observed also from Figure 9, 10 when frame rate is less than or equal 22, it is illustrated that SRTP is little higher than RTP, this due to padding process that influence the video size which was discussed in section 5.2.1.
8. ANALYSIS

The implementation proves the claims discussed in the introduction of this paper as follows:

- The frame rate has an impact on securing video over IP networks.
- The best frame rate is 22 that achieves fewer frame and packet loss, and close to international standard frame rate (26) frame/second for transmission of video over IP networks.
- Our implementation has no extra overheads in the packet format as illustrated in section 3.

8.1 SECURITY ATTACKS

In this subsection we discuss the security aspects of the proposed implementation for different attacks. An encryption scheme is said to be computationally secure if it can withstand the following two criteria. Firstly, the cost of breaking the cipher should exceed the actual value of the encrypted information. Secondly, the time required to break the cipher should exceed the useful lifetime of the information [12]. The proposed implementation model satisfies both the criteria. The security of the AES is checked against cipher text-only, known plaintext as these are the most important attacks.

8.1.1 CIPHER TEXT-ONLY ATTACK

Cipher text-only attack is a model for cryptanalysis where the attacker is assumed to have access to a set of cipher texts and knows the encryption algorithm. This is the most difficult attack since the cryptanalyst has access only to the encrypted data [12]. It is shown that, AES with 10 rounds of encryption and key size of 128 bits, can be broken if the cipher texts are available. Hence, the AES algorithm is secure against cipher text-only attack.

8.1.2 KNOWN-PLAINTEXT ATTACK

Known-plaintext attack, the attacker has access to both the cipher text and plaintext along with the encryption algorithm. The attacker first constructs a partial table with both the plaintext and the corresponding encrypted coefficients from the video collection. Since we used a 128 bit key for encryption he needs to try on an average $2^{127}$ combinations to know which combination of key being used, very difficult to break. Furthermore, the key is randomly generated every time the program accepts new secured connection with clients, making it more difficult for a cryptanalyst to break using known-plaintext attack. Hence the AES is robust against known-plaintext attack also.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed an implementation model that uses TLS protocol as standardized protocol for implementing secure communications over the Internet; rely on public-key cryptography. AES randomly generates a key for encryption for every new secured connection, and we proved that AES has less Encryption time for frame about 1.12ms compared with other related work.

Frame rate has an impact on securing video over IP networks, and we proved that the best frame rate is 22 which achieve fewer frame and packet loss.

All this work may become a core for a security model for transmission video over IP networks, which may be used in manufacture. Our future work will be to reduce the encrypted video size by trying different encryption algorithms hence come up with an ideal video encryption algorithm which takes less encryption time.

It can be extended to use an adaptive algorithm that will choose the best frame rate for transmission depending on the connection speed of client. It can also be extended to videos codec’s like MPEG-4, H.261, and H.264 etc.

REFERENCES


