Design and Testing of two secure Video Conferencing applications based on JMF (Java Media Framework) and VIC (Video Conferencing Tool)*

L. Mengual1, J. Bobadilla2, R. Caballero3, G. Hernández3
DLSIIIS, Facultad de Informática, U.P.M., Madrid, Spain
1{l.mengual, r.caballero, g.hernandez}@fi.upm.es
2{jbobi}@eui.upm.es

Abstract

We present two secure real-time video conferencing solutions with JPEG coding using a cipher engine based on the Blowfish algorithm. One of these solutions is developed from Java Media Framework (JMF) [1]. In this case, our solution consists of encrypting frames coded with JPEG that are later encapsulated in packets of the RTP protocol [2] (Real Time Transport Protocol). Our cipher system is based on the Blowfish algorithm [3] of the Cryptix [4] open source cryptographic software library.

Another solution is developed from VIC (Video Conferencing Tool)[5]. In this case, our solution consists of encrypting the UDP packets that contain the JPEG coded frames incorporated in RTP packets. In this paper we analyse the solutions proposed and test the implementations developed in different data networks (LAN/WLAN, Frame Relay).

Key words: Secure video conferencing systems, JPEG Codec, Blowfish algorithm, LAN/WLAN

1. Introduction

Usage of the Internet for on-line banking, shopping and many others applications like Video Conferencing systems is growing exponentially. Some of these applications involve the transfer of sensitive information such as credit card details. Hence to support these type of networked transactions a number of security techniques have been developed. [7..11].

At the present, the problem of the security in the communications over Internet has extended to real-time communications. The necessity to protect multimedia data distributed over Internet makes it necessary to encrypt the audio/video flows transported on Internet. The aim is that the resulting flow allows its later reproduction in real time. It is therefore very important to choose a specific place to incorporate the cipher engine in the code of our multimedia application, as well as to choose an encryption algorithm fast enough to allow real time communications.

Real time packet transmission should resolve the difficulties inherent in packet networks [12...16], among which we have: the delay and loss of packets, the unordered delivery, the appearance of duplicated packets, etc. Real Time Protocol (RTP) [2] is a protocol based on IP networks that provides end-to-end network delivery services for the transmission of the real-time data. RTP is network and transport-protocol independent, thought it is often used over UDP. RTP enables you identify the type of data being transmitted, determines what order the packets of data should be presented in, and resolve the problem of synchronize media streams from different sources. While RTP does not provide any mechanism to ensure timely delivery or provide other quality of services guarantees, it is augmented by the protocol RTCP (RTP Control Protocol) [2]. It provides control and identification mechanisms for RTP transmissions and enables you to monitor the quality of the data distribution.

Over the past few years, a collaborative effort in the network research community has produced a suite of tools and different systems for multimedia video conferencing over the Internet like JMF [1], VIC [17], Marratech [18], Isabel [19], GnomeMeeting[20], Access Grid [21], MPEG-4IP[22] and others. We have chosen some of the most well known tools, such as JMF and VIC, and we have created real time applications with symmetric-key encryption. In this paper we present the conclusions reached in a comparative study.

Java Media Framework (JMF) is a versatile Application Programming Interface (API) for incorporating time-based media into Java applications and therefore portable and independent of the underlying hardware. Any data that change meaningfully with respect to time can be characterized as time-based media. Such media data can be obtained from a variety of sources, such as cameras, microphones etc. To send or receive a live media broadcast or conduct a Video

---

* This work is supported partially by Science and Technology Ministry of Spain under the projects: “A Multi-Agent System Architecture for the automatic and dynamic Implementation of security protocols (TIC2001-3376)” and “Open System of Multimedia Digital Presentation in Real Time (TIC2000-1734-C03-02)”
Conference over Internet or Intranet, you need to be able to receive and transmit media streams in real time. JMF uses RTP for receiving and transmitting media streams across the network. This way, JMF can: play various multimedia files in a Java applet or application (The formats supported include AVI, MIDI, MPEG, QuickTime, and WAV); play streaming media from the Internet; capture audio and video with your microphone and video camera; process time-based media and change the content-type format; transmit audio and video in real-time on the Internet; broadcast live radio or television programs.

VIC (Video Conferencing Tool) is a real-time video conferencing application over the Internet developed by the Network Research Group at the Lawrence Berkeley National Laboratory in collaboration with the University of California, Berkeley. VIC was designed with a flexible and extensible architecture to support heterogeneous environments and configurations. For example, in high bandwidth settings, multi-megabit full-motion JPEG streams can be sourced using hardware assisted compression, while in low bandwidth environments like the Internet, aggressive low bit-rate coding can be carried out in software.

VIC is based on the Draft Internet Standard Real-time Transport Protocol (RTP) developed by the IETF Audio/Video Transport working group. RTP is an application-level protocol implemented entirely within VIC. You need no special system enhancements to run RTP. Although VIC can be run point-to-point using standard unicast IP addresses, it is primarily intended as a multiparty conferencing application. To make use of the conferencing capabilities, your system must support IP Multicast, and ideally, your network should be connected to the IP Multicast Backbone (MBone).

2. Solutions Presented

In this paper we compare two solutions for real time video transmission with symmetric-key encryption. The Blowfish algorithm is used in both solutions. The two evaluation contexts are: an owner application developed using the API in Java of Java Media Framework, and the VIC, a video conferencing application developed by the Lawrence Berkeley National Laboratory.

In the video conferencing with JMF solution we opted for the implementation of a secure JPEG Codec using the Codec dynamic registration services provided by the JMF API. This means that in this case (see Fig. 1) we designed a JPEG Codec in which the RTP packets have a payload that corresponds to the encrypted video frames using a cipher engine based on Blowfish using the Cryptix library [4].

In the VIC solution we have incorporated the Blowfish algorithm from [6] in the part of the code in charge of constructing the RTP packets and incorporating them in the UDP datagrams. The solution used in this case (see Fig. 2) is therefore not related to the chosen Codec. The most important implementation work involved here was to locate the classes in charge of transporting the RTP packets that contain the video frames, and within them, to encrypt the UDP datagrams generated.

3. Video Conference Application Structure based on JMF

The incorporation of security services to a video conference application, that has been developed with JMF, implies to encrypt the video/audio data that are transported by the network. This circumstance outlines several problems to solve: a first problem resides in the election of an encryption algorithm fast enough to guarantee the real time communications. Another problem resides in choosing the proper location of the cipher engine inside the JMF architecture of classes. Finally, a last problem and, that’s not trivial, is the distribution of the session key among the system users.

Regarding the previous problems our alternative consist in, firstly, selecting the Blowfish algorithm as the responsible cipher engine. Blowfish is a symmetric block cipher of 64 bit block and uses a variable-length key, from 32 bits to 448 bits, fast enough to make the communication in real time possible.

We decide to include the cipher engine in the classes that implement the JPEG video Codec selected. In particular, the location was the classes of the JMF architecture denominated “.packetizer”. The “.packetizer” class is an implementation of the
“BasicCodec” interface inside the architecture of classes and interfaces of JMF. This class encapsulates the JPEG data in RTP packets to be sent through the network. The class “...depacketizer” is invoked in the receiver. This class receives RTP packets in a buffer and depacketizes them to obtain a complete frame in JPEG format. Our solution consists on encrypt the video data before its incorporation to the RTP packets (see Fig. 3).

Figure 3. Secure JPEG CODEC in JMF

The basic structure of our application is composed of the “SecureCustomTrans” class (see Fig. 4). This class, from its method main, obtains the session key in a secure way implementing a security protocol; next it carries out the dynamic registration in the architecture JMF of the secure video JPEG Codec that we have implemented through “registerCustomPayload” method. Once registered the new Codec, the start method of the “SecureCustomTrans” class invokes the methods and functions of the JMF API to establish the RTP session.

public class SecureCustomTrans {
    static Format myJPEGFormat
    static Format myGSMFormat
    private byte[] key;
    
    static boolean registerCustomPayload () {.....}
    private String createProcessors() {...}
    private String createTransmitter() {...}

    private void main(String [] args) {
        java.security.Security.addProvider(new cryptix.provider.Cryptix());
        security_protocol sp=new security_protocol();
        Key = sp.Get_Key ;
    }
    
    registerCustomPayload();
    SecureCustomTrans sct = new SecureCustomTrans (...);
    String result =sct.main(); //main
    }/SecureCustomTrans

Figure 4. SecureCustomTrans class pseudocode

This way the “SecureCustomTrans” class creates two new classes: “CreateProcessor” and “CreateTransmitter”. The first one creates what in JMF is called processor that is the entity in charge of managing the input signal, to choose the appropriate Codec and to return the encoded data. The second class takes charge basically of encapsulating the data that receives from the processor in RTP packets.

The “...jpegencrypted.Packetizer” object (see Fig. 5) was created from the “SecureCustomTrans” class by the “registerCustomPayload” method by means of a constructor that receives as parameter a session key distributed by the security protocol. In this constructor the “CipherEngineCodec” object is created. This will be in charge of encrypting video data with Blowfish algorithm. The kernel of the Codec “...jpegencrypted.Packetizer” is the “process” method that carries out the processing of the input signal through the “inbuf” object and it returns the encryption data through the “outBuf” object.

public class Packetizer extends BasicCodec {
    static final JpegFormat JPEG = new JpegFormat();
    static String
    JPEG_ENCRYPTED = “jpegencryptedtp”, private MotorEncipherationCODEC
    private byte[] mykey;
    public Packetizer (byte[] key) {
        inputFormats = new VideoFormat[]{new VideoFormat(VideoFormat.JPEG)};
        outputFormats = new VideoFormat[]{new VideoFormat(JPEG_ENCRYPTED)};
        mykey = key;
        MyCipherEngine= new CipherEngineCODEC( ); // Packetizer
    }
    public synchronized int process (Buffer inBuffer, Buffer outBuffer){
        byte[] inData = (byte[]) inBuffer.getData();
        System.arraycopy(inData, offset + inBuffer.getOffset(), sincif, 0, copyLength);
        byte[] cifin = MyCipherEngine.BlowfishCipher(sincif, mykey);
    } // Packetizer

Figure 5. “...jpeg.packetizer “ class pseudocode

4. Video Conference Application based on VIC

One of the problems that arose when including security in our video conferencing application based on VIC was the way to add the security. We considered several solutions depending on where the cipher engine was included. One possible solution would be to encrypt at frame level, i.e. every time a frame is obtained from the data source, the source should be encrypted by the application. Another possible solution would be to design a Codec in such a way that it automatically encrypts all the information that goes through it. This Codec will be used to encrypt and decrypt the information depending on whether it is used by the transmitter or the receiver. The last solution to this problem, which was the one that was finally chosen, is to add the security at socket level, i.e. to detect the socket that is being used for the transmission and reception of the data of the corresponding session and to encrypt the data just before it is sent by the transmitter and decrypt it as soon as it is received by the receiver.

To achieve this objective we used a Blowfish cipher engine using the source code [6] as the starting point, which was fast enough to allow it to work in real time. The code that implements and makes the data encryption
possible, using the Blowfish API functions, is incorporated in the Network class methods. The “Network” class (see Fig. 6) incorporates methods that implement the transfer of information between entities through the Windows Sockets interface and it is part of the group of classes invoked from the TclObject class. We have added the symmetrical cipher engine to these methods so that before sending any data it is encrypted with the session key, and as soon as any data is received it is decrypted with the same key.

Figure 6. Class Structure in VIC

5. Performance Study

A wide range of tests were carried out to prove the features of the implementations made. The tests consisted of evaluating the performance of real time video transmission with JPEG code in the two implementations developed, VIC and JMF. The analysis of the transmission was carried out by evaluating the incorporation of a Blowfish cipher engine and by using different Network technologies (LAN 10 -100 Mbps, Frame Relay 2 Mbps and WLAN 11 Mbps). Finally, the possible influence of abrupt movements in front of the camera was studied.

We must point out that in both evaluation environments a fixed RTP packet size of 960 Bytes was used in the VIC environment and 1024 Bytes in the JMF environment. The relative comparison of features was also carried out in the Windows 2000 operating system with a Pentium II processor at 450 MHz with 512 Mbytes on the transmitter and a Pentium II processor at 400 MHz; 256 Mbytes on the video receiver.

5.1 Results in JMF

5.1.1 Cost of frame transmission protocol. Different data networks have been evaluated: LAN (Local Area Networks) 10 -100 Mbps, Frame Relay 2 Mbps and WLAN (Wireless Local Area Networks) 11 Mbps. At JPEG quality 0.2, 14 fps (frames per second) are reached in LAN at 10/100 Mbps or FR at 2 Mbps, whilst only 12 fps are reached in WLAN (see Fig. 7.a). With a JPEG quality of 0.7, 11 fps are reached in LAN at 10/100 Mbps or FR at 2 Mbps, whilst only 8 are reached in WLAN (see Fig. 7.a).

5.1.2 Relationship between the number of frames and the quality without encryption. The set of tests carried out covers a range of JPEG qualities from 0.2 to 0.7, obtaining a clear conclusion from the reduction in the number of frames from a range of 14-12 fps with quality 0.2 to a range of 11-8 fps when the quality is increased. By increasing the quality the JPEG algorithm is compressed to lesser extent and the JPEG frames are larger.

5.1.3 Relationship between the number of frames, the quality and Bits per second without encryption. In direct association with the previous conclusion we can point out the following: At low quality, we have mentioned that more small frames are transmitted due to the strong compression. Therefore, in this situation less bandwidth is required, oscillating between approximately 450 Kbps (WLAN)-500 Kbps (LAN/FR) with quality 0.2 and almost 1250 (WLAN)-1450 Kbps (LAN/FR) with quality 0.7.

5.1.4 Cost of real time encryption. With quality 0.2 we can see that the average rate of fps reached is around 12 fps in LAN/FR whilst in WLAN only 11 fps are reached (see Fig. 7.a). With quality 0.7 we can see that the average rate of fps reached is around 7 fps in LAN/FR and WLAN only 6 fps are reached (see Fig. 7.a).

From the analysis of the results we can deduce that the cost of encryption at low quality (0.2) is approximately 1-2 fps: From 14 fps sent without encryption to 12 fps encrypted in LAN/FR. And from 12 fps sent without encryption to 11 fps encrypted in WLAN. By increasing the quality to 0.7 the cost of using real time encryption is also increased, reaching a difference of 4 fps in LAN/FR:

From 11 fps sent without encryption to 7 fps encrypted. And of 2 fps in WLAN: From 8 fps sent without encryption to 6 fps encrypted.

5.2 Comparison of Results JMF-VIC

In relation to the curves without encryption, the behaviour of VIC and JMF is similar as regards the fps sent. This way, this value increases to 12 fps at medium quality (0.5) with LAN/FR and 10 fps in WLAN, for example (see Fig. 7). The difference is in the bandwidth used from 700 Kbps (WLAN)-750 Kbps (LAN/FR) in VIC and 780 Kbps (WLAN)-900 Kbps (LAN/FR) in JMF.

The number of packets required also varies proportionally: from 90 pps (WLAN)-100 pps (LAN/FR)
in VIC and 110 pps (WLAN)-130 pps (LAN/FR) in JMF. The explanation could be found in the optimisation of the JPEG Codec of VIC as opposed to JMF and the slight difference in the size of the RTP packet used in VIC and JMF.

![Secure JMF (a)](image)

![Secure VIC (b)](image)

**Figure 7. JMF-VIC Comparative test results. WLAN, 400-MHz, CPU**

Nevertheless, the most significant differences are centred on the results obtained using Blowfish cipher engine. Therefore, using VIC the average number of frames sent with medium quality is 10 fps in LAN/FR and 9 fps in WLAN (see Fig. 7.b). In JMF on the other hand, we obtain 8 fps in LAN/FR and 7 fps in WLAN (see Fig. 7.a). The relative difference is therefore 2 fps. Using encryption the use of bandwidth oscillates between 650 Kbps (WLAN) and 700 Kbps (LAN/FR) in VIC and 550 Kbps (WLAN) and 650 Kbps (LAN/FR) in JMF. A clear difference according to the lower number of frames sent by JMF.

VIC and our JMF secure videoconferencing system have been tested using a wireless network (WLAN 802.11b) and a 400-MHz CPU. Fig. 7 shows the test results using different JPEG qualities (y-axis: fps achieved, x-axis: necessary bandwidth). We can observe that the encryption algorithm processing overload affects JMF more than VIC. This is mainly because Java does not run as fast as C/C++; the Blowfish cipher engine runs 3.5 times faster in C/C++ tests than in Java tests (3.03 using a 400-Mhz. CPU and 3.98 using a 1-GHz. CPU).

Both solutions show a similar performance without the use of the cipher engine; however, by using a JPEG quality of more than 0.3, the secure JMF version produces an average of 2 frames per second less than the VIC version. This is the price we have to pay in order to use JMF, a portable, flexible and reliable java-based framework where the new codecs and digital drivers are automatically maintained.

### 6. Conclusions

In this work we have presented two practical real time video conferencing solutions using the Blowfish cipher engine. In the first solution, using JMF we obtain a flexible solution, which, through the dynamic registration of Codecs, allows users to choose the security services in the JPEG Video transmission. In this solution we encrypt the content of the RTP packets. The use of the Java language allows versatility in the video broadcast platform. In the second solution, we propose a cipher engine that has no relation with the Codec in the VIC video conferencing environment. In this solution, the content of the UDP packets that contain the video conferencing frames is encrypted. The cipher engine used developed in C/C++ has been used from [6].

The comparison of results reveals that both solutions present a similar rate of fps without the use of the cipher engine, however there is a greater use of bandwidth in the case of JMF. When using the Blowfish cipher engine the differences are more significant as an average difference of 3 fps is produced in the transmission of JPEG encrypted video. These differences can be attributed, firstly, to the difference in the efficiency of the cipher engine in Java as opposed to in C/C++, as we have already mentioned, and, secondly, to the efficiency of the JPEG Codec in one version or the other.

### 7. References


19. ISABEL: http://isabel.dit.upm.es/

20. GnomeMeeting: http://www.gnomemeeting.org/


22. MPEG-4IP: http://www.mpeg4ip.net/