An end-to-end encryption plugin for video call software

A Thesis Presented

By

WangChong Zhou

to The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

in the field of

Electrical and Computer Engineering

Northeastern University
Boston, Massachusetts

December 2017
# TABLE OF CONTENTS

1  ABSTRACT ................................................................................................................iii

2  INTRODUCTION ........................................................................................................1

   2.1  Background ........................................................................................................1

   2.2  Statement of the Problem ..................................................................................1

   2.3  DirectShow .......................................................................................................2

3  METHOD ....................................................................................................................2

   3.1  System Design ..................................................................................................2

   3.2  Encryption Part .................................................................................................3

   3.3  Decryption Part ...............................................................................................4

   3.4  Controller .........................................................................................................4

   3.5  Image Encryption .............................................................................................5

      3.5.1  Shuffling ....................................................................................................6

      3.5.2  Bitwise XOR ..............................................................................................6

      3.5.3  Block Cipher .............................................................................................6

4  RESULTS ....................................................................................................................7

   4.1  Shuffling ...........................................................................................................7

   4.2  Bitwise XOR ....................................................................................................9

   4.3  Block Cipher ..................................................................................................11

5  CONCLUSION ...........................................................................................................12

6  REFERENCES .........................................................................................................13
1 ABSTRACT

Most of real world video call software lack the end-to-end security for video stream. Encryption is performed during client to service provider to prevent man-in-the-middle attack. However, service providers can easily intercept and record video traffic. In this paper, we implemented a plugin that can add end-to-end security to existing video call software. First, we describe the system design and components of our plugin. Then three different test encryption methods are proposed and compared on our work. Finally, the results of these three methods are evaluated and analyzed. The implemented plugin is proved to be effective.
2 INTRODUCTION

2.1 Background

In traditional instant text messaging, service providers use end-to-end encryption to ensure the privacy of users. End-to-end encryption (E2EE) is a system of communication where only the communicating users can read the messages. In principle, it prevents potential eavesdroppers – including telecom providers, Internet providers, and even the provider of the communication service – from being able to access the cryptographic keys needed to decrypt the conversation. [End-to-end encryption - Wikipedia]

With the evolution of Internet and software technology, more people are using video call software to communicate. Encryption is performed during client to service provider to prevent man-in-the-middle attack. Most of video call software, however, are not building on an end-to-end encryption technique, including Skype and Google Hangout. In the other words, video call service providers can record, change or manipulate the video stream user is sending and receiving. In this paper, we are proposing a technique that can add E2EE security to existing video call software.

2.2 Statement of the Problem

We have the following approaches to do E2EE on video streams:

1. Treat the video stream as normal binary data and apply common stream ciphers upon it.
2. Manipulate the video stream on a per-frame basis, apply image encryption algorithms upon it.

The first approach is easier to implement if we are creating a video call software from scratch and has control of all protocol implementation or network infrastructure. However, this can’t be applied on existing software like Skype. We use the later solution in this work.

Encrypting video stream on per-frame basis, however, is not necessary same as encryption on binary streams. Video call software may have a mechanism that enhance the video quality or adjust the bitrate based on Internet connection speed. In this way, we can’t assure that what we send to video call software on the one side will be same with what we receive on the other side. We will discuss this later in detail.
2.3 DirectShow

Before explaining the design and technique of our work, we will have a short explain on the technologies that our work is built on.

The Microsoft DirectShow application programming interface (API) is a media-streaming architecture for Microsoft Windows. Using DirectShow, applications can perform high-quality video and audio playback or capture. (DirectShow (Windows)) DirectShow was part of DirectX and later became part of Windows SDK. (Microsoft, 2016) DirectShow is based on the Microsoft Windows Component Object Model (COM) framework. It’s an extensible, cross-language and filter-based framework. A filter is a DirectShow components that has one or more pins. Each pin is either used as input or output. The filter either act as a source (has no input pin), or a transform (has both input and output pins) or a renderer (has no output pin).

The only stage we can manipulate video stream is when video call software accessing video streams from cameras. DirectShow provides a set of APIs so that to build a source filter. A source filter is a filter that provide source stream to downstream. In other words, it can act as a camera if we implement all the interfaces that a real camera driver provides.

Using DirectShow as our underlying framework brings us the most widely support of video call software. This includes windows programs using DirectShow natively, Flash programs and programs that uses libraries like ffmpeg or engines like Unity.

3 METHOD

3.1 System Design

Our work is divided into two parts, the encryption part and the decryption part. Each part works as a plugin on a video call software. The following Figure 1 shows how a video stream captured by Alice is encrypted and transported to Bob. The same process runs from Bob to Alice as well. The video call software only sends and receives encrypted video stream.
An end-to-end encryption plugin for video call software

3.2 Encryption Part

The following graph Figure 2 shows the structure of the encryption part.

The source filter we implement here only has one pin, namely the capture pin. This pin is used as a source feed providing video stream for downstream filters.
We build a graph inside the source filter that takes camera input and transform compressed video format into uncompressed format. Camera driver usually compress raw video stream into compressed format. On Windows, it’s usually AVI format. Also, different camera vendors use different color spaces, including RGB and YUV. Therefore, we need this transformation to decompress and standardize video stream ready for encryption. In our work, we transformed each frame into a 24bit RGB BMP image. After that, the source filter does the encryption using the algorithm and key set in its shared memory.

Also, we implement an interface for downstream to query and negotiate dimension, frame per seconds (fps) and color channels. Since each camera hardware is different, we proxy these API calls back to camera driver. One problem of this approach is, a common design of camera driver is to use a same driver to supports several products that use same or similar hardware. Therefore, a driver will support a relatively wide range of output formats and dimensions. While for a particular camera, only one or few formats is actually supported. Therefore, we developed a rule to pick one optimal output format based on what the driver reports.

We will discuss the functionality of Controller later.

3.3 Decryption Part

Since we can’t directly read raw video stream from video call software, the decryptor is fed by capturing the screen of video call software. Then we can decrypt the video frame by frame using same algorithm we selected in encryption. The decrypted bitmap needed to be displayed on the screen. For this purpose, we implemented a standalone GUI controller program.

3.4 Controller

The controller is a standalone GUI program that is used both in encryption and decryption part. It implements the following logic:

1. Read and write source filter’s shared memory. Provide a GUI to control the encryption algorithm and parameters.
2. Provide a GUI container for decryption part.
After the source filter is initialized, it will create a memory space that is mapped readable and writable to other programs. The memory space is spitted into several slots as is shown in Figure 3.

<table>
<thead>
<tr>
<th>1024 bytes</th>
<th>100 bytes</th>
<th>uint16</th>
<th>variant</th>
<th>char[16]</th>
<th>XOR encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct SHRD</td>
<td>pid</td>
<td>width</td>
<td>height</td>
<td>channel</td>
<td>enc_mode</td>
</tr>
</tbody>
</table>

Figure 3

The SHRD struct is used for debug purpose. Shuffle block count and shuffle block mapping is used in the shuffle encryptor below. XOR encryption bitmap is the key used in XOR encryptor below. Bitmap size is equal to the buffer size that the source filter is producing, so that we can do XOR on a bitwise basis. The last 16 bytes of the first 1KB space is used as a block cipher key in the block cipher encryptor below.

3.5 Image Encryption

In our scenario, the encryption part and decryption part is connected through two video call clients. It is common for video call software to adjust the video quality to ensure best video calling experience. Theses adjustment can be divided into at least three aspects:

1. Image resizing or video resampling
2. Image optimization, including sharpening or brightness control
3. Image quality adjustment, the image quality (or video bitrate) is adjusted based on the connection speed and stability.

These three parts all adds a noise to the message, the encrypted video stream. We define noise as unwanted or changed pixel values introduced during transportation. On the decryption side, there will also be noise introduced when we are capturing the screen of video call software. There’re color depth conversion and image resizing. The following Figure 4 shows the diagram of the stage when noise is introduced.
3.5.1 Shuffling

The straightforward method or encryption or obfuscating an image is to split the image into different blocks and then shuffle their orders. For a video with width of 640 pixels and height of 480 pixels, we first split the image into 256 blocks with 40 pixels width and 30 pixels height. Then we use a lookup table to swap these blocks. For a video with dimension of 320 x 240, we generate 256 blocks with dimension of 20 x 15. This method is independent on image size and color depth. Thus, the noise introduced will not be magnified.

3.5.2 Bitwise XOR

We also implemented a XOR based encryption. This method uses a chosen image as key. For encryption, the key image is resized to the same size as plain image and each pixel is XORed. This generates an image with same size as plain image. For decryption, the same key image is used and the encryption process is repeated. This recovers the image to plain image. This method requires resizing, thus the noise will be magnified if we enlarge the image.

3.5.3 Block Cipher

We test 128 bit AES with ECB (Electronic Codebook) mode and 128 bit AES with CBC (Cipher Block Chaining) mode as a block cipher. The BMP format image is encrypted using the cipher and the result is send to downstream as a BMP frame. In ECB mode, each 128-bit block is encrypted using key. Thus, identical plaintext blocks are encrypted into identical cipher text blocks. (Wikipedia) In CBC mode, the
first block is encrypted using key and a given IV (initial vector) and generate a cipher and a IV for next block; all next blocks use the same key and the IV generated in former block. The CBC mode or other feedback-based modes are less susceptible to reply attacks. (Wikipedia) However by using the CBC mode, the noise is significantly magnified because of the chaining design. For ECB mode, each noise pixel only affects the length of block, which is 128bit.

4 RESULTS

In the following test, we are using Skype as the video call software. All the screenshots are captured on receiver’s side. Following is the image we sent without encryption.

![Plaintext image](image)

*Figure 5 Plaintext image*

4.1 Shuffling

The shuffling approach is working as expect. The following images show the result we set the blocks to 4x4 and 16x16.
However, the shuffling is not strictly an encryption. One can still peak the rough image even though it’s shuffled, or use brute force methods to re-shuffle the image. We implement this approach only as a proof-of-working purpose.
4.2 Bitwise XOR

The effect of using a random generated key image on bitwise XOR is shown below. The first sets of images are the result when XORing with doge.jpg.

![Figure 8 XORed with doge.jpg](image1)

![Figure 9 Decrypted using doge.jpg](image2)

The second sets of images are encrypted using a randomly generated image, all pixels values are taken
randomly using a pseudo random number generator.

Due to the noise magnification, some pixels in the key is also represented in the decrypted image. For the first one we can see a rough border of the key we are using. And for the second one, the random pixels are also visible in the decrypted image. The result of encrypted image is totally based on the randomness of key. It’s obviously that if we use a single-colored image as key, the border of original
image will still be visible. There’s another problem even we use a random-generated image as key. If the original image has continuous white (0x00000000 in RGB) or black (0xffffffff in RGB) space, the key bit will be leaked in the cipher text. Thus, this may lead to replay attack.

A solution to this is to generate this random-image key based on time. Or we could also find these white and black space and replace them with random bits.

4.3 Block Cipher

When using the CBC mode, the encrypted image can’t be decrypted. When using ECB mode, the encrypted image can’t be decrypted.

The result for CBC mode is expected and reasonable, as we have stated before. The result of ECB mode is unexpected. We are using 24bit color depth in encryption and decryption, and AES uses 128bit as a block. Since 128bit block can’t be divided by 24bit, this may result to a pixel being separated into different blocks. But we can improve this by converting the color depth to 32bit. This will be our future work on this project.

The following images shows the encrypted and decrypted image using AES-128-ECB mode. The key is set to 128 bits of 0’s.

![Figure 12 Image encrypted using AES-128-ECB](image-url)
5 CONCLUSION

In this paper, we implemented an end-to-end encryption plugin for existing video call software. We first analyze the background and environment of our problem. Secondly, we describe the design of this plugin and make a theoretical reasoning for its effectiveness. Then we implement three encryption methods in our work and test upon Skype. Among those, the simple shuffling and XOR method is working as expected. The cipher image encrypted by block cipher method can’t be properly decrypted. But we also propose a way of improving block cipher mode.
6 REFERENCES


