國立暨南國際大學資訊工程學系

碩士論文

以即時資訊隱藏技術改進網路通話 G.711 編碼之安全機制研究

Improved VoIP Security with Real-time Speech Hiding in G.711

指導教授：吳坤熹博士

研究生：王鐘逸

中華民國九十七年二月

論文名稱：
以即時資訊隱藏技術改進網路通話 G.711 編碼之安全機制研究
校院系：國立暨南國際大學資訊工程所　　頁數：54
畢業時間：97 年 2 月　　學位別：碩士
研究生：王鐘逸　　指導教授：吳坤熹博士

中文摘要

在即時溝通系統中，即時音訊隱藏指的是將秘密音訊藏入掩護音訊。藉由將秘密音訊藏入掩護音訊中，我們可以得到一個聽起來有意義並且與掩護音訊相同的隱匿音訊。因此，即使攻擊者在網際網路上擷取到音訊封包，他仍然無法察覺到內容含有秘密音訊。本篇論文提出了一種即時音訊隱藏的方法，應用在 Voice over Internet Protocol (VoIP)的即時通訊系統上。

G.711 是在 VoIP 裝置中最被廣為支援編碼技術；我們針對 G.711 提出了一種新的即時音訊隱藏設計。於 Scheme 1 實驗顯示此演算法的執行時間為 0.257 毫秒，這個結果證明此方法在即時 VoIP 應用上是適用的。我們進一步導入 Advanced Encryption Standard（AES）加密演算法以提高系統的安全性，在 Scheme 2 的實驗中需要 0.28 毫秒的執行時間，也同樣顯示此方法適用於即時 VoIP 應用。

關鍵字：資訊隱藏, 即時音訊, AES, G.711, Linphone, Speex, VoIP

Title of Thesis：

Improved VoIP Security with Real-time Speech Hiding in G.711

Name of Institute：Department of Computer Science and Information Engineering, National Chi Nan University Pages：54

Graduation Time：2/2007 Degree Conferred：Master

Student Name：Chungyi Wang Advisor Name：Quincy Wu

Abstract

Speech hiding is a powerful information protection mechanism in real-time communication systems. By hiding one secret speech into the cover speech, we can get a stego speech, which sounds meaningful and indistinguishable from the original cover speech. Therefore, even if attackers intercept the audio packets on Internet, they would not notice that there is another speech hidden inside those packets. In this thesis, we propose a scheme for speech hiding in a real-time communication system such as voice over Internet Protocol (VoIP).

We propose a novel design of real-time speech hiding for G.711 codec, which is widely supported by almost every VoIP device. Two schemes were proposed in this thesis. Experimental results in Scheme 1 show that the running time for the proposed algorithm takes only 0.257ms, which is suitable for real-time VoIP applications. By adding AES encryption to enhance the security, we obtain Scheme 2 whose running time is 0.28ms, which also shows that this scheme is suitable for real-time applications.

Keyword： AES, G.711, information hiding, Linphone, real-time speech, Speex, steganography, VoIP

Table of Contents

Figure Index

Table Index

1. Introduction

1.1 Motivation

In voice over Internet Protocol (VoIP) systems, human voice needs to be encoded as digital packets, and transmitted over Internet. Since Internet is an open environment where packets may be eavesdropped by malicious attackers, it is a common approach to protect the audio contents by an encryption algorithm, such as the Data Encryption Standard (DES) or the Advanced Encryption Standard (AES) [1]. However, there may be some potential problems for these approaches.

Although encryption can protect the contents of the message, the speech packets will sound meaningless and chaotic after they are encrypted. The attackers could thus easily notice the speech is protected by encryption. This encourages the attackers to invest more resource to decipher the messages, because encrypted messages usually contain valuable secret inside it. Therefore, compared with cryptographic techniques that only conceal the contents of information, the approach of *information hiding (steganography)* tries to hide not only the contents but also their own existence [2]. This technique generally choose some cover messages which contains no sensitive information, and embeds the secret information into the cover messages. Information hiding has been widely adopted in protecting messages in plaintext [3], audio files in WAV or MP3 formats [4], and image files with BMP (bitmap) [5] or compressed JPEG format [6] .

In this thesis, we propose a scheme for information hiding in real-time VoIP systems. We hide the secret speech into one meaningful cover speech, and obtain a stego speech which can be transported over public Internet. Ideally, a good hiding algorithm will produce a stego speech which sounds almost the same as the original cover speech. As the difference is indiscernible to the observers, it would be difficult for attackers to notice it.

For human auditory system (HAS), generally little distortions of the voice data will not be noticed, and the speech can still be understood easily. We define the *speech hiding space* as the range in which a voice packet is allowed to be distorted while keeping indistinguishable to human ears. Generally, in a sampled voice data, modifying the Least Significant Bits (LSBs) will cause the minimum distortion, so the hiding space is usually chosen in LSBs.

There is another problem: attackers could possibly decipher secret speeches

successfully after analyzing large amounts of audio packets. Therefore, adaptive solutions have to be proposed to further protect the VoIP conversation.

1.2 Related work

In [3], the possibility is investigated to hide information steganographically in the "noise" created by automatic translation of natural language documents. It uses the phenomenon of errors created by automatic translation to embed secret messages. Here is the basic idea: the sender and the receiver jointly select one public text, and translate it to get cover message. Then, the sender encodes secret with cover message to obtain stego message. The receiver will compare the cover message and the stego message, and then obtaining the hidden data from the differences. However, for an attacker, the differences in stego messages just seem plausible. Because different translation packages may generate minor errors in the translated sentences, it should be difficult for an adversary to determine if these errors come from the steganography or inaccuracy of different translation packages.

Information hiding in image applications also attracted many studies. The JPEG format image could also support steganography. In [6] it proposed a method to hide secret messages by modifying *comment marker*, which is a basic component in the JPEG file structure. There were two methods proposed in [6]. The first method is called replacement methodology, which loads comment marker and modifies the contents. It maintains the original size of the cover image. In the second method, insertion methodology hides secret messages into the cover image which had no comment marker. These two methods could embed information into a cover image and generate a stego image which is discernible from the original image.

In [4], it mentions that most audio information hiding schemes are not music-based. The dynamic ranges of HAS are much higher than human visual system (HVS). Therefore, the size of hiding data in audio information hiding would be much less than image format. In [4] it proposed a method to solve this problem by utilizing the dynamic range filter in MP3-format. Because filtered signals are mostly outside the HAS dynamic ranges, it will be difficult for human to notice that this approach is being applied..

In [7] it proposed a scheme in information hiding based upon sub-band. By observing that the little distortions in high frequency band would not be noticed in HAS, it splits cover speech into two frequency bands (called low and high). After embedding compressed secret speech into the high frequency band, combining these two bands together and sending it to the receiver. It also applies encryption algorithms

on compressed secret speech. In other words, even if the adversary intercepted the message, they could not decipher the secret speech without correct keys.

1.3 Solution

This thesis proposes two different schemes. In Chapter 2, we propose Scheme 1 which selects hiding samples from the cover speech with interval 1 and modifying the least significant r bits. Modifying parameter r will create different size of hiding space, i.e., the number of bits that will be chosen in LSBs to store the hidden information. However, care must be taken in determining the size of the hiding space. If too many bits are chosen, the modified speech sample will have lots of noises, and thus will become easy to be noticed that information hiding approaches are applied.

Unfortunately, if attackers understand the algorithm of Scheme 1, they could reconstruct the secret speech. For this reason, we propose Scheme 2 in Chapter 4 to solve this potential problem. This new scheme adopts encryption algorithms to prevent attackers from getting the contents of the secret speech.

We will describe the detailed procedures in the following chapters. Chapter 2 will introduce the flow chart of Scheme 1, then analye the speech quality and processing time in different value of r in Chapter 3. The flow chart of Scheme 2 will be described in Chapter 4, and the speech quality and processing time of this scheme will be analyzed in Chapter 5. The last chapter will mention conclusion and the future work.

2. Scheme 1

In this thesis, we study the information hiding scheme with G.711 [8] as the codec of the cover speech. G.711 is the most popular codec which is supported in every VoIP and circuit-switching system [9]. Its sample rate is 8000Hz, where every sample is encoded to 8 bits. It will output one 64kbps audio stream. There are two versions of G.711: the A-law and the μ-law. The μ-law is used in North America and Japan, and A-law is used in Europe and the rest of the world. They both support good quality with Mean Opinion Score (MOS) value 4.3, which is much higher than other audio codecs such as G.723, G.726, and G.729.

G.711 algorithm takes a 16-bit linear audio sample as input and converts them into 8 bits. As shown in Figure 2.1, for every 160 samples with each sample consisting of 16 bits will be encoded into 160 samples with 8 bits.

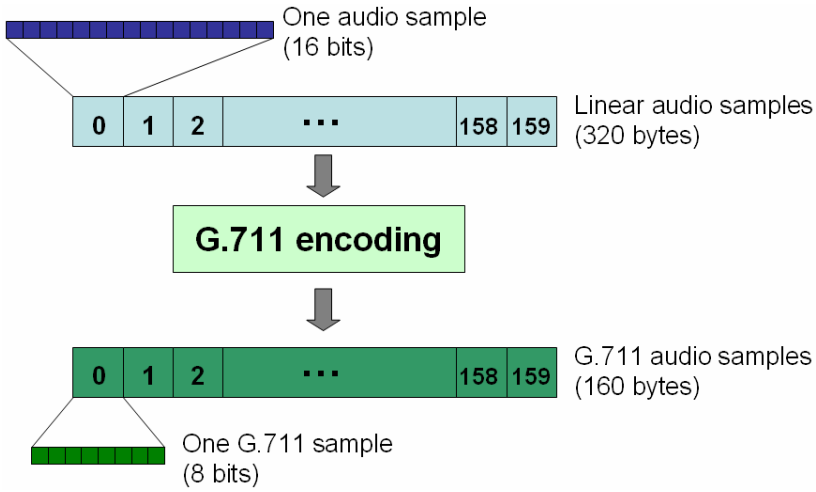

Figure 2. 1 G.711 encoding

Next, as shown in Figure 2.2, for a sender to apply the information hiding technique to protect the voice stream, a speech hiding space is allocated from a speech packet. The speech packet is generally called cover speech and denoted by C. The speech hiding space is denoted as HS and will be used to store the secret speech. Let us denote the secret speech as S. Suppose we obtain speech S' by compressing S, and hide them into HS. The new speech packet (which contains S') is called the stego speech, which is denoted by G. Now the stego speech G can be sent to the receiver via public Internet.

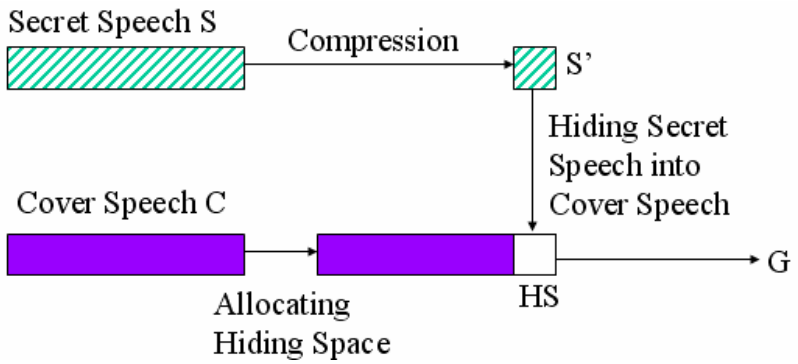

Figure 2. 2 Flow chart of the sender

Figure 2.3 shows the flow chart of the receiver. After the receiver receives the stego speech G, he/she can extract S' from HS, and decompress S' to get the secret speech D. (Please note that we choose a different notation because D may differ from the original secret speech S, because of the lossy nature of the compression algorithm.)

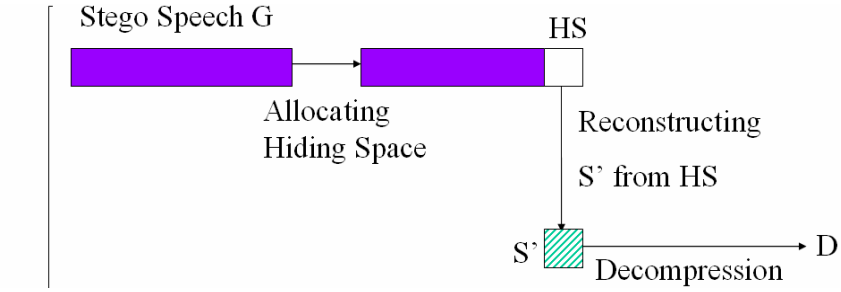

Figure 2. 3 Flow chart of the receiver

2.1 Speech Hiding Space

Determining the size of speech hiding space is very important. The space is used for hiding secret speech so that attackers could not easily notice that there is one secret speech hiding in stego speech. To achieve this goal, the space should cause only slight distortions and the noises should be evenly distributed into all speech packets. Moreover, to prevent the cover speech from being modified dramatically during the information hiding process, we choose only half of cover speech samples to hide secret data. Suppose there are N samples in one speech packet, and denote the number of every sample by 0,1,2…(N-1), where N is an even integer. Among them, the hiding samples are those numbered by 1,3,5,7…(N-3),(N-1), as shown in Figure 2.4.

For every hiding sample, we choose the least significant r bits to store the secret speech. By doing so, we can keep the distortion to the cover speech as low as possible, and thus the noises caused by the slight distortion would not be perceived by human ears. These r bits in the hiding samples will collectively be used to store the secret

speech, as shown in Figure 2.4.

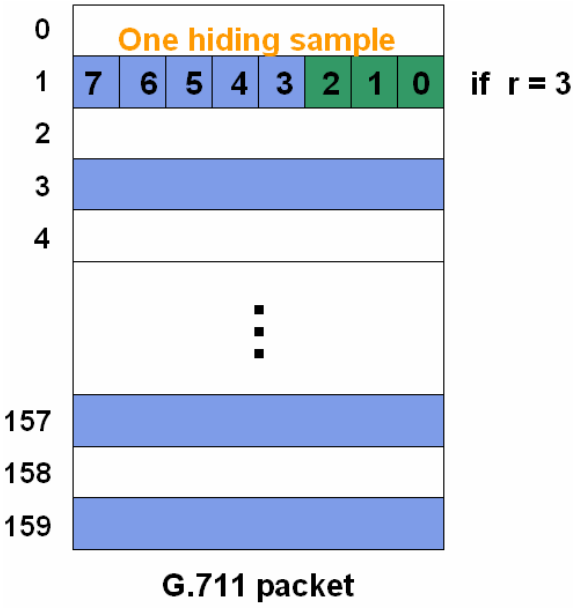

Figure 2.4 Allocating hiding space

2.2 Compressing Secret Speech

To reduce the noises caused by the steganographic approach, the size of hiding space must be much smaller than the cover speech. Therefore, the secret speech must be compressed before it is inserted into the cover speech. Furthermore, to achieve the requirement of real-time playing, the compression algorithm must match the limitation of short running time. Here we choose the speech compression tool Speex [10] to compress the voice data.

Speex is an open source compression software. It is based on Code Excited Linear Prediction (CELP) [11] and is designed to compress voice at bit-rates ranging from 2 kbps to 44 kbps. It supports compressing modes like narrowband (8 kHz), wideband (16 kHz), and ultra-wideband (32 kHz) in the same bit-stream. It also supports echo cancellation and noise suppression. We assume the default number of samples is 160 for a G.711 μ-law packet. Therefore, we choose the narrowband mode in Speex, where the input data are exactly 160 samples with each sample consisting of 16 bits.

To utilize Speex API [12], we need to specify the quality of compression. Table 2.1 shows the sizes of the output frame for each required quality (assume the input data consist of 160 speech samples).

Table 2. 1 Speex compression quality and the output frame size

Quality	**FrameSize (bytes)**
0	6
1	10
2	15
3	20
4	20
5	28
6	28
7	38
8	38
9	46
10	62

Let Ns denote the size of the compressed secret speech, and Nc denote the number of samples in the cover speech, then the formula for calculating the size of hiding space r can be written as:

$$r = \lceil Ns * 8/(Nc/2) \rceil \quad (1)$$

Ns*8 stands for the total number of bits of S', and (Nc/2) stands for the total number of hiding samples (Scheme 1 only chooses half of samples in the cover speech, so Nc/2 = 160/2 = 80).

The value of r should depend on the required quality of the compressed speech. As shown in Figure 2.5, if the required quality of compression is 6, then according to Table 2.1, Ns would be 28 bytes. Therefore, r can be calculated by formula (1),

$$r = \left\lceil \frac{N_s * 8}{\frac{N_c}{2}} \right\rceil = \left\lceil \frac{224}{80} \right\rceil = 3.$$

On the other hand, if the required quality is 1, Ns would be 10 bytes, so the value of r is 1 bit, from a similar calculation.

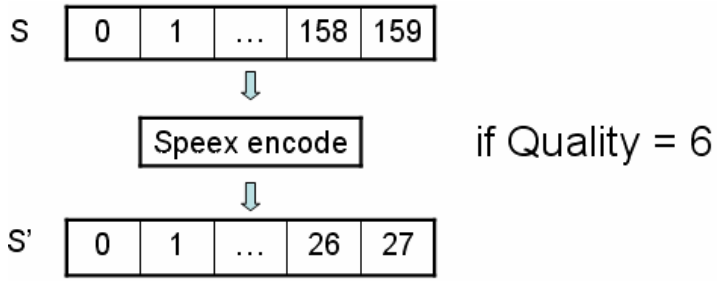

Figure 2. 5 Speex Compressing

2.3 Hiding Secret Speech into Cover Speech

Figure 2.6 illustrates the process for hiding the secret speech into the cover speech. The sender compresses the secret speech and embeds it into the cover speech in the hiding space. Because we select half of samples in a cover speech as the hiding space, sample #1, #3, #5, … #159 will be utilized to store the secret. For each one, we serially pick up r bits from S', and place them into the LSBs of the corresponding hiding sample.

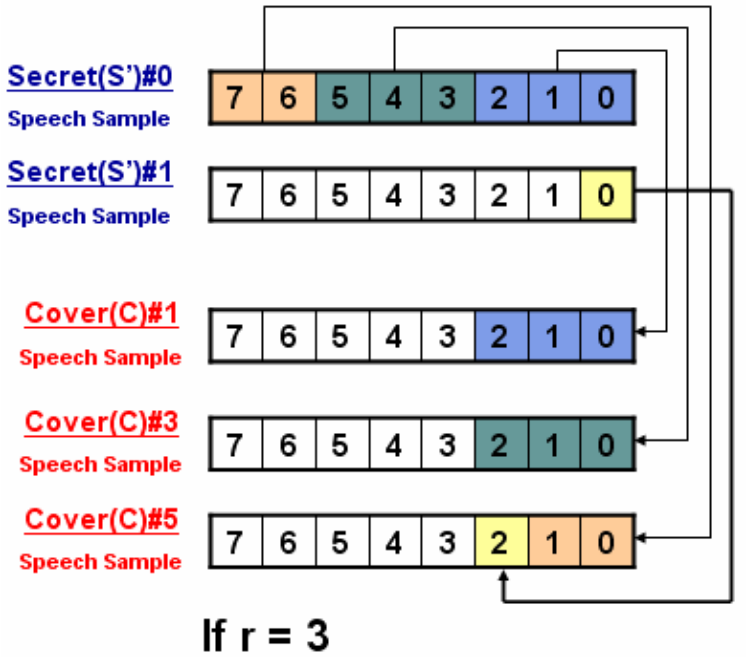

Figure 2. 6 Embedding Secret Speech into Cover Speech

2.4 Extracting Secret Speech from Stego Speech

With prior knowledge of the value of r, the receiver extracts r bits from the hiding space (there are 80 such samples in each packet), and concatenates them to reconstruct S'.

The next step is decompressing S' to obtain the secret speech D. Because Speex is a lossy compression algorithm for voice, there may be a little difference between the secret speech D and the original secret speech S, but the distortion is acceptable and the difference should be indiscernible by human ears.

2.5 Linphone implementation

To verify the performance of this algorithm, we implemented this real-time speech hiding scheme in Linphone. To verify the empirical performance, our proposed algorithm is implemented in Linphone [13], which is an open source VoIP software. It can send/receive audio, video and instant messages. It supports audio codecs including G.711, GSM and iLBC. It also supports video codecs including H263-1998, MPEG4 and theora. Linphone complies with the Session Initiation Protocol (SIP) and is able to interoperate with most SIP-compatible phones and SIP proxy servers.

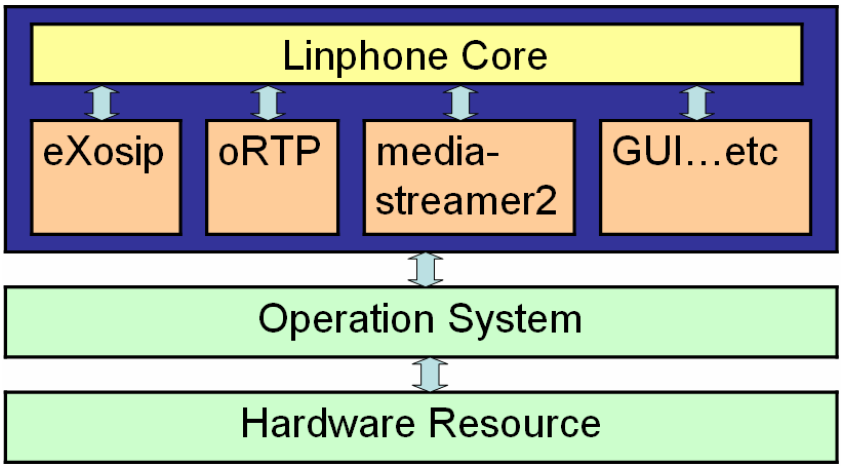

Figure 2. 7 System Components of Linphone

Linphone consists of several components as illustrated in Figure 2.7: The **oRTP** module will utilize Real-time Transport Protocol (RTP) to prepare the sampled audio data in RTP packets. The **eXosip** is a library based on the GNU oSIP protocol stack, which handles the signaling for call setup and teardown in SIP. The **mediastreamer2** is one important part of linphone. It contains several objects to process audio and video data and outputs them to the oRTP module or a local file. Moreover, it contains codec objects to compress audio and video.

It is necessary to understand how Linphone handles a VoIP call. As shown in Figure 2.8, when both parties agree on the choice of audio codec, Linphone will create corresponding data structures to process the voice coding and decoding. The structures of codecs are defined in the library of mediastreamer2, which supports a variety of codecs, including G.711, GSM and other optional codecs. The mediastreamer2 also creates the structures which process the input and output of audio. For example, the input from a microphone and the output to speaker are

handled by the mediastreamer2.

Moreover, mediastreamer2 also handles the task of packing speech data into Real-time Transport Protocol (RTP) packets. It will create the structures to process sending and receiving in RTP. In other words, in Linphone software, mediastreamer2 is an important component which handles all tasks about audio speech processing.

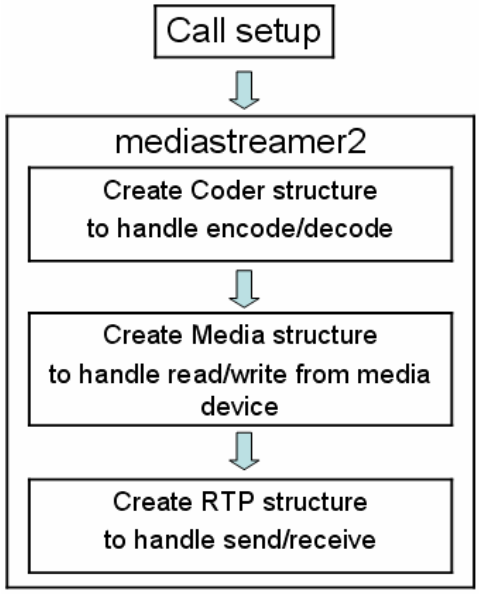

Figure 2. 8 Call setup process of mediastreamer2

2.5.1 Call Setup in mediastreamer2

Suppose we denote the structure of reading audio array as ACs, encoding audio array as CEs, and sending RTP packet as RSs. When one call is set up, the mediastreamer2 will use the function audio_stream_start_full(…) in mediastreamer2/src/audiostream.c to create the six structures mentioned above. Then we assign ACs to CEs, which will encode the audio array from ACs. Next we assign CEs to RSs, then RSs will pack the audio array which is encoded by CEs to send.

Reversely, we denote the structures of writing audio array by MPs, decoding audio array by CDs and receiving RTP packet by RPs. First we assign MPs to CDs , and assign CDs to RRs.

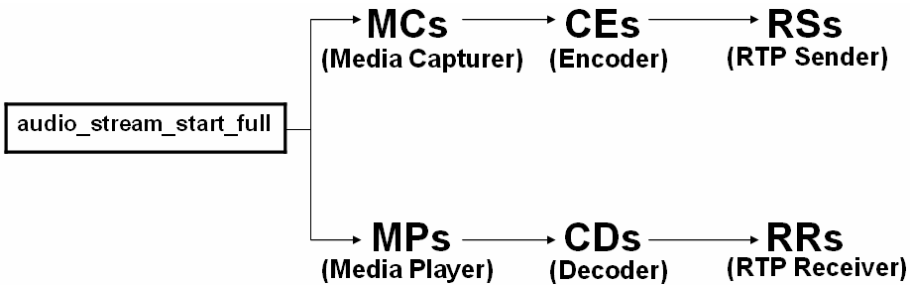

Figure 2. 9 Calling sequence of audio_stream_start_full(…)

By the flow chart of sending speech, MCs reads audio array from the soundcard. Next CEs reads the audio array from MCs and encodes it, and CEs will write the audio array which is encoded into the queue of RSs for sending. In the final step RSs will pack the audio array into a RTP packet and transmit it over Internet. When the receiver gets the packet, it will store it into RRs. RRs reads the audio array from the packet, then CDs will read and decode it. Next CDs writes the audio array which is decoded into MPs for playing. In the final step MPs will write the audio array into the soundcard to play.

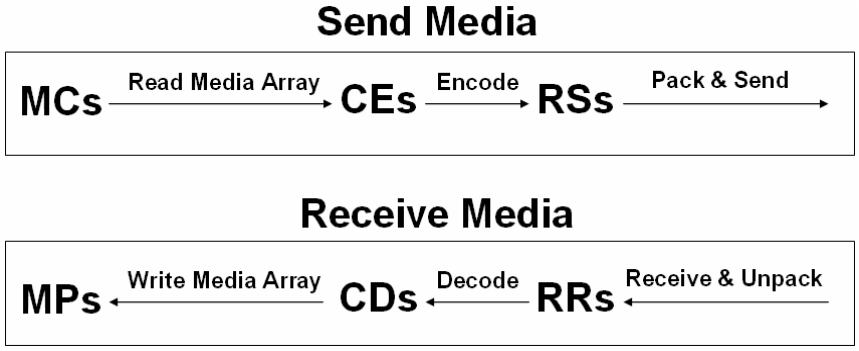

Figure 2. 10 Data flow of mediastreamer2

Our proposed scheme focused on modifying codecs to support information hiding, so we only need to modify the functions in Linphone which processes codec. For codec G.711 μ-law, mediastreamer2 will create two structures to process μ-law encoding and decoding, respectively, and use the functions in mediastreamer2/src/ulaw.c to finish the task.

The functions in ulaw.c for encoding and decoding are ulaw_enc_process(…) and ulaw_dec_process(…), respectively. In ulaw_enc_process(…), the function reads the audio array from MCs and encodes it, then write it into the queue of RSs for sending. In ulaw_dec_process(…), the function reads the audio array from RRs and decodes it, then write it into the queue of MPs for playing.

Let us denote the audio array by A, and the encoded audio array is denoted by A', The flow chart of the ulaw.c can be illustrated by Figure 2.11.

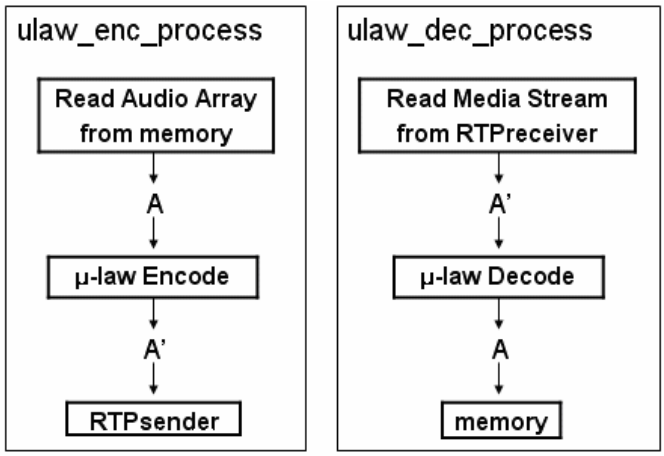

Figure 2. 11 Data flow of ulaw.c

2.5.2 Modifying Codes

In ulaw_enc_process, the audio array should be processed by the flow chart of Figure 2.2 before it was written to the RTPsender. By the flow chart of Figure 2.5, 160 samples will be read from the secret speech, and then compressed to obtain S'. Then picking the hiding speech space HS from the cover speech C. Here we use the encoded audio array A' as the cover speech. In the last step, as depicted in Figure 2.6, S' is hidden into HS and the stego speech G is obtained.

In ulaw_dec_process(…), the flow chart is in the reverse direction with ulaw_enc_process(…), so we need to modify the code before decoding. We extract HS from G before decoding, and reconstruct the secret speech S', and decompress S' to obtain D.

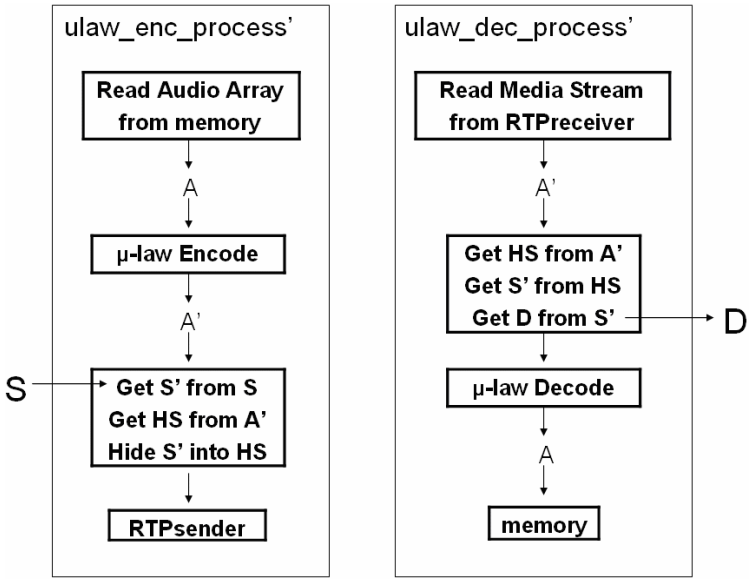

Figure 2. 12 Modifying the data flow in ulaw.c

2.6. G.711 μ-law and A-law

The coding algorithms of μ-law and A-law are very similar, except that A-law algorithm provides a slightly larger dynamic range than the μ-law at the cost of worse proportional distortion for small signals. This means that our proposed scheme could be implemented in both G.711 μ-law and A-law.

For the A-law, there is a file alaw.c in mediastreamer2 to handle it. There are two functions to handle encoding and decoding which are called alaw_enc_process(…) and alaw_dec_process(…), respectively. The flow chart of these two function are almost the same as Figure 2.11, and all modifying steps could follow the steps shown in Figure 2.12.

3. Quality Analysis of Scheme 1

Determining the quality of compressing is a very important issue is this scheme. The value of Signal-to-Noise Ratio (SNR) will be better if higher quality is chosen, but the size of secret speech also becomes bigger according to Table 2.1. It would cause the value of r to increase, and make the speech significantly distorted. With the increased noises, the attackers might easily notice the existence of the hiding speech.
On the other hand, if lower quality of compressing is chosen, the secret speech D will get worse value of SNR. However, the smaller value of r makes the stego speech G indistinguishable from the cover speech C, so the attackers could not notice the hiding speech easily.

In this section, we will describe two methods with different parameter of r. Then show the experimental results for two different quality levels in compression, and the SNR for these cases.

3.1 Scheme 1 (r=3) and Scheme 1 (r=1)

In Scheme 1 (r=3), the quality of compressing in Speex is 6, and the Ns is 28-bytes. We assume the size of speech packet is Nc=160. According to formula (1), we can get the value of r is 3.
In Scheme 1 (r=1), the quality of compressing in Speex is 1, and the Ns will be 10-bytes. We also assume the size of speech packet is Nc=160. According to formula (1), we can get the value of r is 1.

3.2 Signal/Noise Ratio

We ran the experiment in an isolated network, to make sure that there is no background traffic interfering the experiment. The same cover speech is utilized in all testing. First, we test the original G.711 μ-law speech (without information hiding) and obtain the value of SNR is 5.237dB. This result will be compared with other experimental results.

The cover speech is a 8.07MB audio file in WAV format, which is sampled at the rate of 44.1kHz and quantized in 16 bits. The length of this WAV file is 48 seconds. The measured value of SNR is 8.55dB. (All the SNRs in this subsection is obtained

by repeating the cover speech 8 times, i.e. totally 48*8=384 seconds, and calculating the average.)

The size of secret speech is a 12.8MB audio file in WAV format, which is sampled at the rate of 8kHz and quantized in 16 bits; the length of this the secret speech is 274 seconds. The measured value of SNR is 4.22dB.

We consider the following three scenarios to measure the SNR value of the encoded signals.

(a) We encode this cover speech by G.711 μ-law. The measured value of SNR is 5.237dB. This will be the base for comparing the following scenarios.

(b) We measure the stego speech G generated by the Scheme 1 (r=3), the value of SNR is 4.596dB. Compared with (a), the noises are higher and they are easy to be noticed by the attackers. The quality of compressing in Speex is 6, every 160 samples with 16-bits sample-size will be compressed into 28-bytes, so the compression ratio is 320:28=11.43:1. The secret speech D is decompressed from the stego speech; its value of SNR is 3.647dB.

(c) We measure the stego speech G generated by the Scheme 1 (r=1), the value of SNR is 4.964. Compared with (b), the noises are lower and more difficult to be noticed. The quality of compressing in Speex is 1, every 160 samples will be compressed into 10-bytes, so the compression ratio is 320:10=32:1. The secret speech D is decompressed from the stego speech; its value of SNR is 3.358dB. Although the quality of secret speech here is lower, the noises of stego speech is also lower than Scheme 1 (r=3), which makes it less discernible

The SNR value and compression ratio of different methods are shown in Table 3.1.

Table 3. 1 Compression ratio and SNR of different encoding methods

Method	Compression Ratio	SNR of D (dB)	SNR of G (dB)
None	-	-	8.55
G.711μ-law	-	4.22	5.237
Scheme 1 (r=3)	11.43:1	3.647	4.596
Scheme 1 (r=1)	32:1	3.358	4.964

3.3 Processing Time

In addition to the voice quality, we also want to make sure the performance of this scheme can meet the real-time requirement of VoIP. Let us assume that the default internal to send G.711 speech packets in Linphone is 20ms. In other words, every speech packet must be encoded or decoded in 20ms, otherwise it will be dropped.

Our experiment is running on a Pentium 4 machine with 3.4GHz CPU and 512MB memory, with Linux Fedora Core 6 as the operating system. We ran 8000 iterations (and calculate the average) to measure the process time of each component for encoding, decoding, compressing, and decompressing. The encoding and decoding time in μ-law are 0.003ms and 0.001ms, respectively. For the sender in the Scheme 1 (r=3), the compressing time is 0.249ms, and the time for storing the secret speech is 0.005ms. Therefore, the total time required for the sender is 0.249ms+0.005ms+0.003ms=0.257ms.

For the receiver, the decompressing time is 0.036ms, and the time for extracting the secret speech is 0.003ms. Therefore, the total time required by the receiver is 0.036ms+0.003ms+0.001ms=0.04ms.

For the sender in the Scheme 1 (r=1), the compressing time is 0.252ms, the time for storing the secret speech is 0.002ms. For the receiver, the decompressing time is 0.04ms, the time extracting the secret speech is 0.001ms. Therefore, the total time of the sender is 0.252ms+0.002ms+0.003ms=0.257ms. Similarly, the total time of the receiver is 0.04ms+0.001ms+0.001ms=0.042ms.

Table 3. 2 Running time of Scheme 1 which (r=3) and (r=1)

Operation	Scheme 1 (r=3) (ms)	Scheme 1 (r=1) (ms)
S->S' (Send)	0.249	0.252
Pick HS & Hide S' (Send)	0.005	0.002
μ-law (Send)	0.003	0.003
Total (Send)	0.257	0.257
Pick HS & Reconstruct S' (Recv)	0.003	0.001
S'->S (Recv)	0.036	0.04
μ-law (Recv)	0.001	0.001
Total (Recv)	0.04	0.042

3.4 Summary

We test the stego speech G by the Scheme 1 (r=3), and then obtain the value of SNR is 4.596dB. According to the result of experiment, the noises increase by 12%, this makes big difference from the original speech. If an attacker catches the speech packets, he could easily notice the noise, and then detects that there is secret information inside stego speech.

Although the quality of secret speech decompressed by Speex in this method is better, it introduces more noises into the stego speech. This contradicts with our original goal for hiding speech.

The experimental environment is the same as Scheme 1 (r=3). We obtain the value of SNR for this method is 4.964dB. According to the result of experiment, the noises increase by 6%, which is not perceivable to human ears.

Although the quality of secret speech in this method is lower, it introduces fewer noises into the stego speech. This result shows that the Scheme 1 (r=1) can achieve our goal to confuse the attackers in speech hiding.

As shown in Figure 3.1, no matter the algorithm is implemented using Scheme 1 (r=3) or (r=1), the delay it introduced is shorter than 1ms, so this result shows that the proposed scheme is suitable to be applied in real-time VoIP systems.

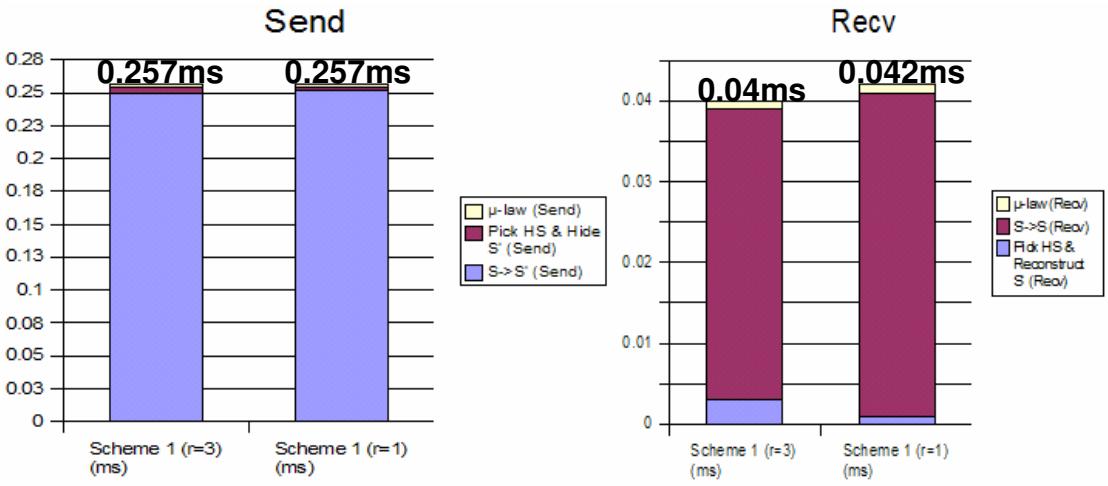

Figure 3. 1 Runtime of Scheme 1 which (r=3) and (r=1)

4. Scheme 2

In Scheme 1 (r=1), stego speech will increase 6% noise from cover speech. It is good enough to achieve the goal that hiding secret speech but not be noticed. Furthermore, the runtime in Scheme 1 (r=1) is less than 1 ms both in sender and receiver. This shows that it would be normally done in real-time system.

However, there is the same potential problem both in Scheme 1 (r=1) and (r=3). When attackers know the hiding space is allocated by LSB after they used data analysis, moreover, they could use correct decompressing algorithm to obtain secret speech. In other words, Scheme 1 would not provide enough protection as we expected. We can't guarantee that attackers would not understand every step in this algorithm. Furthermore, we even need to assume that attackers will understand the algorithm in detail. By the way, we should propose corresponding scheme for avoiding this risk in Scheme 1.

The better way is to apply encryption algorithm. After compressing the secret speech to get S', the next step is encrypting S' to get encrypted speech S''. In the final step, S'' will hide into C to get G. If attackers don't have secret key to decrypt S'', they cannot get secret speech even if they successfully reconstructing S''. The scheme not only keeps the characteristic which is not easily noticed by attackers, but also has information security by adopting encryption algorithm.

The second problem in Scheme 1 is that it can only hide static audio speech (e.g. WAV audio file with G.711 encoding). Scheme 1 can't provide the capability for the purpose which a real-time calling needs protection. For this requirement, Scheme 2 can set the cover and the secret speech by the device or the static audio speech as we wish. In hiding static audio speech, the device speech (e.g. microphone) would be the cover speech. In hiding real-time speech, the static audio speech would be the cover speech.

Figure 4.1 shows the flow chart of Scheme 2 in sender. The first step is to set the cover and the secret speech, this is decided by user. In real-time calling situation, the device speech will be the secret speech and the static audio speech will be the cover speech. After compressing secret speech to get S', for ensuring security of this algorithm, the next step is to encrypt S' to obtain the encrypted speech S''.

In the final step, S'' will be hided into HS to get the stego speech G, then sending G to the receiver.

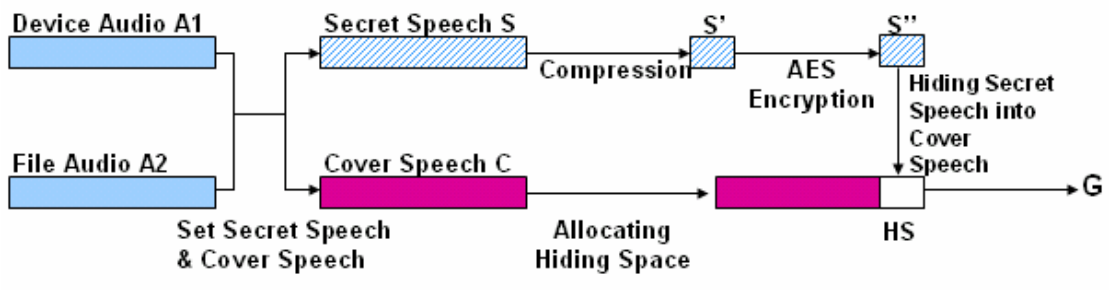

Figure 4. 1 Flow chart of sender in Scheme 2

Figure 4.2 shows the flow chart of Scheme 2 in receiver. After receiving the stego speech G by the receiver, the first step is allocating the hiding space HS. Next step is to reconstruct S" from HS, then decrypting S'' to get S' by the same key which the sender used for encrypting. The last step is to decompress S' to obtain the secret speech D for playing.

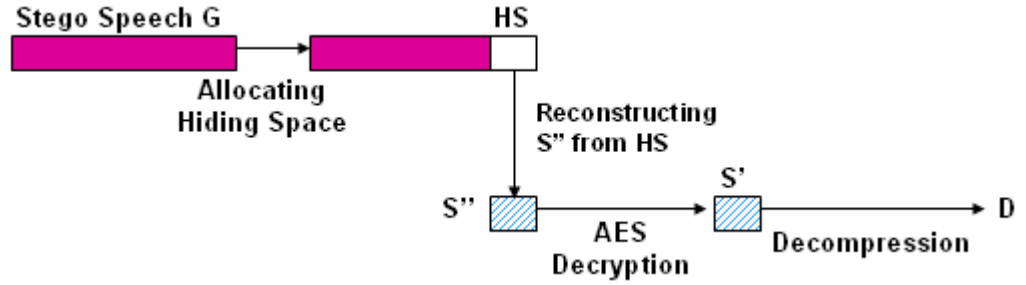

Figure 4. 2 Flow chart of receiver in Scheme 2

4.1 AES Encryption/Decryption

In Scheme 2, the Advanced Encryption Standard (AES) is adopted for encrypting the secret speech. AES is a block cipher adopted as an encryption standard by the U.S. government. It replaces the Data Encryption Standard (DES) and widely using in the world.

AES was announced by National Institute of Standards and Technology (NIST) as U.S FIPS PUB 197 on November 26, 2001, and then it becomes standard on May 26, 2002. The cipher was developed by Joan Daemen and Vincent Rijmen, so the AES also known as "Rijndael" from the portmanteau of the names of the inventors.

AES is fast both in software and hardware, and it has little memory requirement. AES has a fixed block size in 128 bits and the key size could be 128, 192, or 256 bits. (i.e. 16, 24, 32 bytes) AES operates on a 4x4 array of bytes and the initial value is a block of plaintext. There are four steps in one encryption round, AddRoundKey, SubBytes, ShiftRows and MixColumns. [14]

Many public products use the 128-bits key which is enough for normal usage.

But in TOP SECRET information, either 192 or 256 key lengths will be required for using. [1]

After introducing the AES, selecting adaptable length of key is very important. The main factor of key length selecting is the size of hiding space. In Scheme 1, hiding samples are picking up with interval one from the cover speech and modifying the least significant r bit(s) as part of hiding space.

In Scheme 1 (r=3), modifying the least significant 3 bits for hiding. The total size of hiding space is 28 bytes. However, the louder noise would be noticed by attackers. In Scheme 1 (r=1), modifying the least significant 1 bits for hiding. The total size of hiding space is 10 bytes. This scheme causes little distortions from the original speech, so modifying the LSB is a better way for allocating hiding space.

The minimal key length of AES is 16bytes. By the way, this scheme must provide hiding space bigger than 16 bytes. Scheme 1(r=1) can only provide hiding space by 10 bytes, but this space is not enough for AES encryption.

To break this limitation, Scheme 2 uses new rule for selecting hiding samples. It keeps the attribute of modifying the LSB in Scheme 1 (r=1), but changed the selecting hiding samples without interval. By this way, the size of hiding space is increased by 2 times. In other words, it is 20 bytes. This result shows the size of hiding space matches the requirement of minimal AES key length with 16 bytes.

4.2 Speech Hiding Space

As section 4.1 mentions in allocating the hiding space before, every cover speech sample is used for hiding with modifying the LSB. The total size of hiding space is 20 bytes by section 4.1. There are 16 bytes for hiding the secret speech, and the remaining 4 bytes is used for storing parameters of negotiation in sender and receiver. Here we denote these 4 bytes as the *control area*.

Figure 4.3 shows the allocation of the hiding space. In these 20 bytes, #4 to #19 is called the *data hiding area*. This area is used for hiding secret speech whatever is encrypted or not. #0 to #3 is the control area. #0 mainly is used for signaling control, #1 for storing parameter and #2 to #3 is reserve for future using.

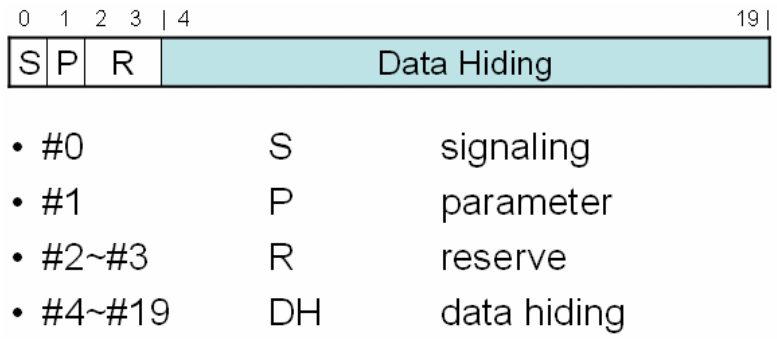

Figure 4. 3 Allocation of Hiding Space

Here we describe in detail of the control area. Every bit in #0 has different usage for signaling. As Figure 4.4, #0 bit(0) means that if this packet using speech hiding or not. If #0 bit(0) is 1, then whole flow chart of Scheme 2 is not executed. If #0 bit(0) is 0, then Scheme 2 is executed. It includes allocating the hiding space, compressing and hiding the secret speech into the cover speech.

#0 bit(1) means that encryption algorithm is executed or not. If the value is 0, S' is not encrypted. If it is 1, S' will encrypt to obtain S'' before hiding into C. The remaining 5 bits (2~7) is reserved for future using.

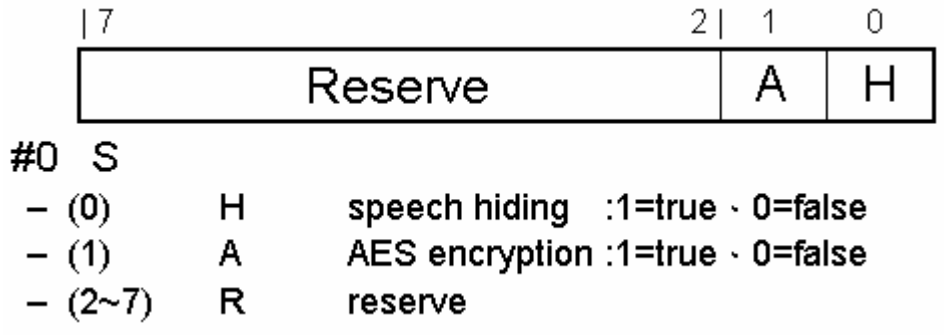

Figure 4. 4 Allocating of #0

When #0 bits(1) is 1, the #1 will store the AES round. There are two proposes for recording the AES round. First is to decrypt correct secret speech D in receiver by current round, second is to recover errors by packet loss. In the Internet, packet loss situation can't be fully avoided and should be carefully consider in this scheme. In other words, the AES round should be included in Scheme 2.

As Figure 4.5 shown, #1 is 1 byte, this means the total rounds can be recorded are 256. The value range of #1 is 0 to 255. #1 is increasing by 1 after one speech packet encrypted. It will reset to 0 when #1 is bigger than 255. In other words, Scheme 2 allows packet loss with maximal numbers of 256. This value could be enough in Scheme 2. For example, if every G.711 packet which is sanded by 20 ms, in the worst case, packet communicating has been terminated longer about 5 seconds. In other words, this call may be ended due to unexpected situation.

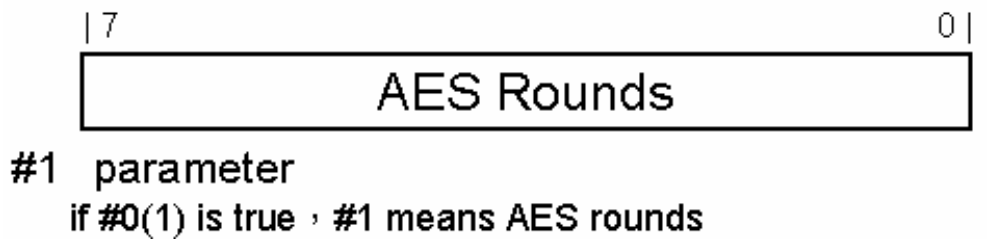

Figure 4. 5 Allocating of #1

4.3 Compressing Secret Speech

Secret speech should be compressed before AES encrypting. To use the libspeex as section 2.2 mentioned, the compressing quality must be set. According to the AES key length by 16 bytes which Scheme 2 selected, the quality should be set to 2 according to the Table 2.1. The size of output data is 15 bytes. The selecting quality not only can match the requirement of minimal key length by 16 bytes, but also getting better quality of D than Scheme 1 (r=1).

4.4 Encrypting the Compressed Secret Speech

Figure 4.6 shows the relation of S', S" and HS. For AES encrypting in 16 bytes, the compressed speech needs to add one padding byte. The padding byte sets to 0. After AES encrypting, S" is hided into the data hiding area of HS. Because of encryption is executed, #0 bit(1) is set by 1 and #1 stores the AES round in HS before sending G to the receiver.

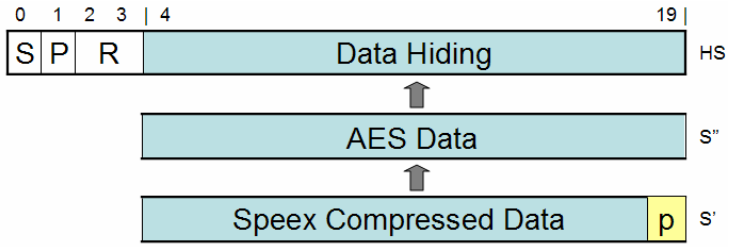

Figure 4. 6 Relation of S', S" and HS

4.5 Decrypting the Encrypted Secret Speech

In the receiver, the secret speech will be reconstructed from HS in G, but the receiver still can't be sure that this secret speech is encrypted or not. According to the Figure 4.7, the first step is to check both #1 bit(0) and #1 bit(1) in HS is true. If they

both are true, then receiver can realize that whole flow chart of Scheme 2 and the AES encryption is executed.

The second step is to check if the system AES round is equal to the packet AES round. Here we denote the system AES round by Sr and the packet AES round by Pr. Sr is recorded both in sender and receiver, and the Pr is exactly the value of #1 in HS.

If AES encryption is executed, the next step is to do one-round decryption and increase Sr by one. Checking if Pr is equal to Sr for being sure that the secret speech is correct by AES decrypting. If it is false, backing to do one-round decryption and increase Sr by 1 till the value of Sr is equal to Pr.

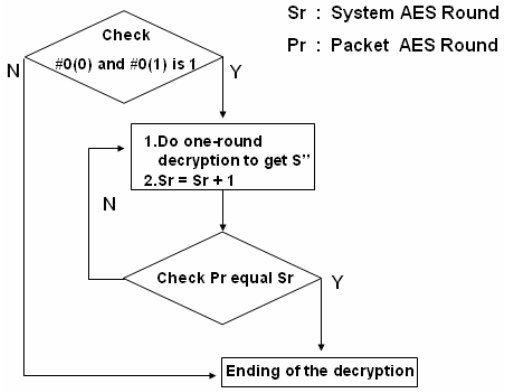

Figure 4. 7 Flow chart of decrypting

4.6 Linphone implementation

We also implement this real-time speech hiding scheme in Linphone to verify the performance of the algorithm. As Figure 4.8 in ulaw_enc_process shown, the first step is to set the secret speech and the cover speech. The speech source could be the device audio array A or the static audio array Af. The second step is compressing S to get S'. Next, encrypting S' to get S''. In the end, hiding S'' into the cover speech to obtain G.

On the other hand, there is an additional decryption step after reconstructing S'' from A' in ulaw_dec_process. The decryption should follow the flow chart of section 4.5; the receiver will acquire correct secret speech D to play.

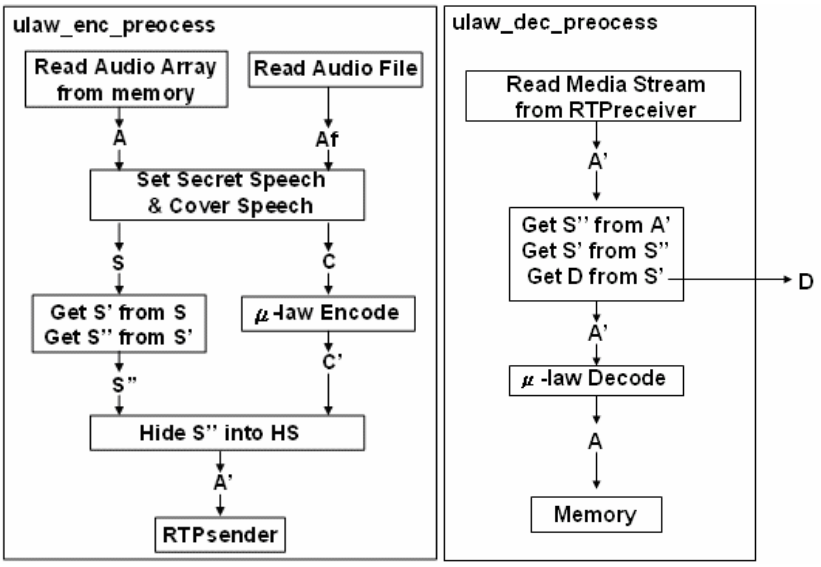

Figure 4. 8 Modifying the data flow in ulaw.c

To implement the AES encryption and decryption in Figure 4.8, we choose a library of AES. The library is built by Brian Gladman, the algorithm code can be used in C/C++ and Pentium family assembler [15]. The most advantage of the library is simply implementing AES, and all codes are very light. It is the reason which why we choose the library in the scheme. To adopt this library, the first step is to include the aes.h header file. The function aes_enc_blk(…) can process encrypting, and the function aes_dec_blk(…) can decrypt reversely.

Here we make a header file aesmatt.h to handle AES processing. For a start, aesmatt.h should be included in ulaw.c. The function aesinit(…) must be processed both in sender and receiver. In ulaw_enc_process, aesencode(…) is added for processing AES encrypting. In ulaw_dec_process, aesdecode(…) is added for handling one-round or multi-round decrypting for packet loss recover.

4.7 G.711 μ-law and A-law

As section 2.6 mentioned before, G.711 μ-law and A-law are very similar in coding algorithm. To implement A-law in Linphone, two functions need to be modified in alaw.c. These two functions are alaw_enc_process(…) and alaw_dec_process(…). The flow chart of them also could be referenced in Figure 2.11, and all modifying steps could follow the steps shown in Figure 4.8.

5. Quality Analysis of Scheme 2

5.1 Signal/Noise Ratio

We also ran the experiment in an isolated network. To compare the result from Scheme 1, the same cover and secret speech in chapter 3 are utilized in this experiment.

The measured value of SNR of stego speech in Scheme 2 is 5.017dB. The quality of compressed secret speech is 2, every 160 samples with 16-bits sample-size will be compressed into 15-bytes, so the compression ratio is 320:15=21.33:1. The value of SNR of secret speech D is 3.454dB.

The SNR value and compression ratio of Scheme 1 (r=1) and Scheme 2 are shown in Table 5.1.

Table 5. 1 Compression ratio and SNR of different encoding methods

Method	Compression Ratio	SNR of D (dB)	SNR of G (dB)
None	-	-	8.55
G.711μ-law	-	4.22	5.237
Scheme 1 (r=1)	32:1	3.358	4.964
Scheme 2	21.33:1	3.454	5.017

5.2 Processing Time

We also want to ensure the performance of Scheme 2 can meet the real-time requirement of VoIP. Every speech packet also must be encoded or decoded under 20ms, otherwise it will be dropped.

Our experiment is running on the same machine and operating system from section 3.2. We also ran 8000 iterations (and calculate the average) to measure the process time of each component for encoding, decoding, compressing, decompressing, encrypting, and decrypting. The encoding and decoding time inμ-law are also 0.003ms and 0.001ms, respectively.

For the sender in Scheme 2, setting the cover and the secret speech needs 0.002ms, the compressing time is 0.25ms, the encrypting time is 0.021ms, and the time for storing the secret speech is 0.004ms. Therefore, the total time required for the

sender is 0.002ms+0.25ms+0.021ms+0.004ms+0.003ms=0.28ms.

For the receiver in Scheme 2, the decompressing time is 0.038ms, the decrypting time is 0.005ms, and the time for allocating HS and reconstructing S" is 0.002ms. Therefore, the total time required for the receiver is 0.038ms+0.005ms+0.002ms+0.001ms=0.046ms.

Table 5. 2 Running time of Scheme 1 (r=1) and Scheme 2

Operation	Scheme 1 (r=1) (ms)	Scheme 2 (ms)
Set C and S	-	0.002
S->S' (Send)	0.252	0.25
S'->S" (Send)	-	0.021
Pick HS & Hide S" (Send)	0.002	0.004
μ-law (Send)	0.003	0.003
Total (Send)	0.257	0.28
Pick HS & Reconstruct S" (Recv)	0.001	0.002
S"->S' (Recv)	-	0.005
S'->S (Recv)	0.04	0.038
μ-law (Recv)	0.001	0.001
Total (Recv)	0.042	0.046

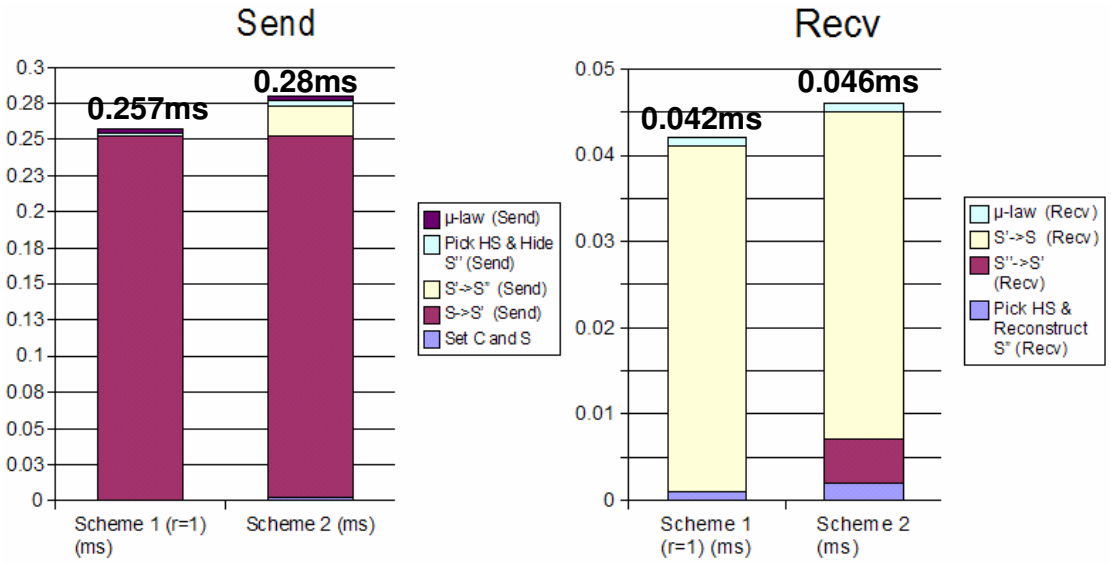

Figure 5. 1 Runtime of Scheme 1 which (r=1) and Scheme 2

5.3 Summary

The SNR value of Scheme 2 is close to Scheme 1(r=1) and makes little distortion from the original speech. This experimental result shows that Scheme 2 also has the advantage which is not easily noticed by attackers.

The total required time both in the sender and the receiver are shorter than 1ms, so this result shows that Scheme 2 is also suitable to be applied in real-time VoIP system.

In addition, Scheme 2 can provide protection of secret speech even if attackers successfully reconstructed S", in other words, Scheme 2 can provide more applicable security than Scheme 1.

6. Conclusion and Future Work

In this thesis, we proposed a scheme for real-time speech hiding. By hiding the secret speech into the cover speech, which sounds like a normal audio stream, it would prevent eavesdroppers from knowing the existence of the secret speech. This would increase the security of VoIP systems when the voice is transported over Internet, which is an insecure channel where network packets may be eavesdropped. By applying Speex to compress the secret speech, we are able to hide it in a smaller space. The choice of quality in the compression will result in different compression size.

In our experiment of Scheme 1, choosing higher quality as level 6 requires 3 bits in a sample packet to be replaced, and thus makes the noise notable (SNR=4.596dB). To make it less obvious, lower quality as level 1 was chosen, and only 1 bit in each sample packet will be replaced. This proved to be a better approach, with SNR=4.964dB, which is superior to the previous experiment where 3 bits were allocated to store the secret speech.

In Scheme 2, all samples are selected to hide the secret speech to get larger space. The value of SNR in Scheme 2 (5.017dB) is close to the value of Scheme 1 (r=1). This result shows that Scheme 2 can keep the advantage of speech hiding as shown in Scheme 1. By applying AES encryption algorithm after compressing the secret speech, we could further enhance the security of the secret speech.

Current VoIP applications generally incorporate multimedia channels, including audio, video, and text. With successfully speech hiding in real-time VoIP applications, naturally it would be interesting to investigate the possibility to apply similar techniques to real-time image hiding or text hiding in a multimedia session to improve the overall security on the communication system in the future.

Reference

1. CNSS Policy No. 15, Fact Sheet No. 1, National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information , June 2006 [http://www.cnss.gov/Assets/pdf/cnssp_15_fs.pdf]
2. F.A.P. Peticolas, R.J. Anderson, and M.G. Kuhn, “Information Hiding – A Survey”, IEEE Trans. Proc. Thy, Vol. 87, No.7, pp. 1062-1078, July 1999.
3. Christian Grothoff, Krista Grothoff, Ludmila Alkhutova,Ryan Stutsman, and Mikhail J. Atallah. “Translation-based steganography”. In Proceedings of Information HidingWorkshop (IH 2005), pages 213–233. Springer-Verlag, 2005.
4. Bao, P.and Xiaohu Ma, “MP3-resistant music steganography based on dynamic range transform”, IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS 2004), 18-19 Nov. 2004, pp.266-271.
5. EasyBMP [http://easybmp.sourceforge.net/steganography.html]
6. Dorian A. Flowers, “Investigating Steganography”, ACM-SE 42, April 2004
7. Chin-Chen Chang, Richard Char-Tung Lee, Guang-Xue Xiao, Tung-Shou Chen “A new speech hiding scheme based upon subband coding”, IEEE Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia.
8. ITU-T Recommendation G.711. Pulse Code Modulation for voice frequencies, Nov. 1988.
9. Daniel Collins, Carrier Grade Voice over IP, 2nd Ed., McGraw-Hill, September 2002.
10. Speex [http://www.speex.org/]
11. J.P. Campbell, Jr., T.E. Tremain and V.C. Welch, The Federal Standard 1016 4800 bps CELP voice coder, Digital Signal Processing 1:145–154 (1991).
12. Speex Reference Manual 1.2-beta2 [http://www.speex.org/docs/api/speex-api-reference.pdf]
13. Linphone [http://www.linphone.org/]
14. FIP PUB 197 : the official AES standard [http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf]
15. Implements of AES (Rijndael) in C/C++ and Assembler, Brian Gladman [http:// fp.gladman.plus.com]

Appendix

Appendix 1. Codes of Scheme 1

Linphone/mediastreamer2/src/ulaw.c

…

```
//msfilter.h is modified for Scheme 1
#include "mediastreamer2/msfilter.h"
#include "g711common.h"

//Including Scheme 1 related file
#include "g711matt.h"
#include "speexmatt.h"

/*
g711matt.h+
        |-encode1(...)      Allocating HS and Hiding S' into C (r=3)
        |-encode9(...)      Allocating HS and Hiding S' into C (r=1)
        |-decode1(...)      Allocating HS and Reconstructing S' from G (r=3)
        |-decode9(...)      Allocating HS and Reconstructing S' from G (r=1)

speexmatt.h+
            |-speex_encode_data(...)      Decompressing S' to get D (r=3)
            |-speex_decode_data(...)      Decompressing S' to get D (r=1)
            |-...
*/

//audiostream.c is modified for Scheme 1
```

…

```
static void ulaw_enc_process(MSFilter *obj){
```

…

```
while (ms_bufferizer_read(bz,buffer,size_of_pcm)==size_of_pcm){
        mblk_t *o=allocb(size_of_pcm/2,0);
        int i;
        //*o->b_wptr means the array which is encoded by G.711 mu-law
        //(int16_t*)buffer means the array
        //              which is loaded from device (microphone)
        for (i=0;i<size_of_pcm/2;i++){
                *o->b_wptr=s16_to_ulaw(((int16_t*)buffer)[i]);
                o->b_wptr++;
        }

        //Declaring variables
        char * hidden;
        int h_size;

        //Loading audio array from file and compressing by Speex to get S'
        speex_data_output(hidden, obj);

        //Allocating HS and hiding S' into HS by Scheme 1
        switch(obj->g711matt_mode_enc) {
            //Scheme 1 (r=3)
            case 1:
                    h_size = 28;
                    hidden = (char *)(malloc(h_size));
                    encode1(o, size_of_pcm/2,hidden, h_size);
                       break;
            //Scheme 1 (r=1)
            case 9:
                    h_size = 10;
                     hidden = (char *)(malloc(h_size));
                     encode9(o, size_of_pcm/2,hidden, h_size);
        }
}

mblk_set_timestamp_info(o,dt->ts);
dt->ts+=size_of_pcm/2;
ms_queue_put(obj->outputs[0],o);
```

```
}

...

static void ulaw_dec_process(MSFilter *obj){
        mblk_t *m;
        while((m=ms_queue_get(obj->inputs[0]))!=NULL){
                mblk_t *o;
                msgpullup(m,-1);
                o=allocb((m->b_wptr-m->b_rptr)*2,0);

                for(;m->b_rptr<m->b_wptr;m->b_rptr++,o->b_wptr+=2){
                        *((int16_t*)(o->b_wptr))=ulaw_to_s16(*m->b_rptr);
                }

                //Declaring variables
                int size = 160;
                char * samples;
                short * hidden = (short *)(malloc(320));

                //Allocating HS and reconstruting S' by Scheme 1
                switch(obj->g711matt_mode_dec) {
                        //Scheme 1 (r=3)
                        case 1:
                                samples = (char *)(malloc(30));
                                decode1(m, size, 1);
                                break;
                        //Scheme 1 (r=1)
                        case 9:
                                samples = (char *)(malloc(29));
                                decode9(m, size, 7, samples);
                                break;
                }

                if(obj->g711matt_channel_dec == 1) {
                    //Decompressing S' to get D
                        speex_dec_data(obj, samples, hidden);
```

```
            //Setting D back into speech array which will play by device(player)
            o->b_wptr-=(2*size);
            for(i=0;i<size;i++,o->b_wptr+=2){
                *((int16_t*)(o->b_wptr))=hidden[i];
            }
        }

        freemsg(m);
        ms_queue_put(obj->outputs[0],o);
    }
}

…
```

Linphone/mediastreamer2/include/mediastreamer2/msfilter.h

…

```
struct _MSFilter{
    MSFilterDesc *desc;
    /*protected attributes */
    ms_mutex_t lock;
    MSQueue **inputs;
    MSQueue **outputs;
    MSFilterNotifyFunc notify;
    void *notify_ud;
    void *data;
    struct _MSTicker *ticker;
    /*private attributes */
    uint32_t last_tick;
    bool_t seen;

    //Scheme 1

    //encode by (r=3) or (r=1)
    int g711matt_mode_enc;
    int g711matt_mode_dec;
```

```
        //0:normal channel
        //1:secret channel
        int g711matt_channel_dec;

        int g711matt_state_enc;
        int g711matt_state_dec;

      //0:play cover speech one time
      //1:repeating play cover speech
        int replaymode;

        //cover speech file keeper
        FILE * fin;
        char * filename;

      //wav file header struct
        void * wh;

      //index of playing wav file
        int file_index;

        //struct which is need in speex encoding
        void * g711_enc_state;
        SpeexBits g711_enc_bits;

      //struct which is need in speex decoding
        int speex_dec_needInit;
        void * speex_dec_state;
        SpeexBits speex_dec_bits;
};
```

Linphone/mediastreamer2/src/audiostreamer.c

…

```
#include "command_line.h"
```

```
/*
command_line.h+
                        | -command_loop(...)       Loading user commands
*/

...

AudioStream * audio_stream_start_full(RtpProfile *profile, int locport, const char
*remip,int remport, int payload,int jitt_comp, const char *infile, const char *outfile,
MSSndCard *playcard, MSSndCard *captcard, bool_t use_ec)
{
      ....

      //Set encode/decode mode in Scheme 1
      //1:Scheme 1 (r=3)
      //9:Scheme 1 (r=1)
       stream->encoder->g711matt_mode_enc = 9;
       stream->decoder->g711matt_mode_dec = 9;
       stream->decoder->g711matt_channel_dec = 0;

      //Wav file of Cover speech related
       stream->encoder->replaymode = 1;
       stream->encoder->fin = NULL;
       stream->encoder->filename = "/root/hidden.wav";

      //Init encode/decode state
       stream->encoder->g711matt_state_enc = ENC_STATE_NONE;
       stream->decoder->g711matt_state_dec = DEC_STATE_NONE;

      //Loading user command by pThread
       hidden_controler * hc = (hidden_controler *)(malloc(sizeof(hidden_controler)));
       hc->g711enc = stream->encoder;
       hc->g711dec = stream->decoder;
       pthread_t pid;
       pthread_create(&pid, NULL, command_loop, (void *)hc);

       //Speex decode Init setting
       stream->decoder->speex_dec_needInit = 1;
```

```
        return stream;
}
...
```

Appendix 2. Codes of Scheme 2

Linphone/mediastreamer2/src/ulaw.c

…

```
//msfilter.h is modified for Scheme 2
#include "mediastreamer2/msfilter.h"
#include "g711common.h"

//Including Scheme 2 related file
#include "g711matt.h"
#include "speexmatt.h"
#include "aesmatt.h"

/*
g711matt.h+
        |-encode9(...)          Allocating HS and Hiding S' into C
        |-decode9(...)          Allocating HS and Reconstructing S' from G
        |-setParameter(...)     Setting #0~#3
        |-getParameter(...)     Getting #0~#3
        |-...

speexmatt.h+
        order_mode(0)
        |-speex_data_output(...)          Loading audio array from wav file
                                          and compressing S to get S'

        order_mode(1)
        |-speex_init(...)                 Init Speex
        |-speex_read_data(...)            Loading audio array from wav file
        |-speex_encode_data_part(...)     Compressing S to get S'

        |-speex_dec_data(..)              Decompressing S' to get D
        |-...

aesmatt.h+
```

```
                |-aesinit(...)        AES encryption/decryprion Init
                |-aesencode(...)    AES encoding
                |-aesdecode(...)    AES decoding
*/

…

static void ulaw_enc_process(MSFilter *obj){

        …

        while (ms_bufferizer_read(bz,buffer,size_of_pcm)==size_of_pcm){
            //Declaring variables
             int i;
             char * hidden;
             int h_size = 16, h_dis = 1, h_start = 32;
             hidden = (char *)(malloc(h_size));

             //order_mode
             //0:Using wav file for cover speech
             if(obj->order_mode == 0) {
                 //Loading audio array from wav file
                 // and compressing S to get S'
                  speex_data_output(hidden, obj);
             }
             //1:Using device(microphone) for cover speech
             else if(obj->order_mode == 1){
                 //If this is first time to run the ulaw_enc_process(...)
                 //     , open the audio file pointer
                   if(obj->g711matt_state_enc == 0) {
                         OpenFIN(obj);
                         speex_init(obj);

                         obj->g711matt_state_enc = 10;
                   }

                   short * file_clip = (short *)(malloc(320));
                   //Loading audio array from wav file
```

```
        speex_read_data(file_clip, obj);

        //Exchanging device memory array & file_clip
        //Copying device memory array to temp array
        int temp[160];
        for(i=0;i<160;i++)
                temp[i] = ((int16_t*)buffer)[i];

        //Setting file audio array to device memory array
        for(i=0;i<160;i++)
                ((int16_t*)buffer)[i] = file_clip[i];

        //Copying temp array to file_clip
        for(i=0;i<160;i++)
                file_clip[i] = temp[i];

        //Compressing S to get S
        speex_encode_data_part(hidden, file_clip, obj);
}

//AES encryption
//For getting S" from S'
if(obj->aes_enc_needInit)
        aesinit(obj, 1);
if(obj->needAES)
        aesencode(hidden, obj);

mblk_t *o=allocb(size_of_pcm/2,0);
for (i=0;i<size_of_pcm/2;i++){
        *o->b_wptr=s16_to_ulaw(((int16_t*)buffer)[i]);
        o->b_wptr++;
}

//Allocating HS
if(obj->needHidden) {
        switch(obj->g711matt_mode_enc) {
                case 9:
                        h_size = 16;
```

```
                        encode9(o, size_of_pcm/2, hidden, h_size, h_start, h_dis);
                        break;
                }
        }

        //Setting #0~#3
        setParameter(o,obj,size_of_pcm/2);

        mblk_set_timestamp_info(o,dt->ts);
        dt->ts+=size_of_pcm/2;
        ms_queue_put(obj->outputs[0],o);
    }
}

…

static void ulaw_dec_process(MSFilter *obj){

    mblk_t *m;
    while((m=ms_queue_get(obj->inputs[0]))!=NULL){
        mblk_t *o;
        msgpullup(m,-1);
        o=allocb((m->b_wptr-m->b_rptr)*2,0);

        //Declaring variables
        int size = 160;
        char * samples;
        short * hidden = (short *)(malloc(320));

        //getParameter to set #0~#3
        getParameter(m, obj);

        for(;m->b_rptr<m->b_wptr;m->b_rptr++,o->b_wptr+=2){
            *((int16_t*)(o->b_wptr))=ulaw_to_s16(*m->b_rptr);
        }

        //Allocating HS and reconstruting S' by Scheme 2
        switch(obj->g711matt_mode_dec) {
```

```
                case 9:
                        samples = (char *)(malloc(16));
                        decode9(m, size, 32, 1, samples);
                        break;
        }

        //AES decryption
        //For getting S' from S"
        if(obj->aes_dec_needInit)
                aesinit(obj, 0);
        if(obj->needDeAES)
                aesdecode(samples, obj);
      //aesdecode(...) will looping do one-round till Sr == Pr

      //Checking Sr == Pr
      //If true, than decompressing S" to get secret speech D
        if((obj->aes_dec_round-1) == obj->aes_dec_round_read) {

                if(obj->g711matt_channel_dec == 1) {
                   //Decompressing S" to get secret speech D
                        speex_dec_data(obj, samples, hidden);

                        o->b_wptr-=(2*size);
                        for(i=0;i<size;i++,o->b_wptr+=2){
                                *((int16_t*)(o->b_wptr))=hidden[i];
                        }
                }

        }

        freemsg(m);
        ms_queue_put(obj->outputs[0],o);

    }
}

…
```

Linphone/mediastreamer2/include/mediastreamer2/msfilter.h

...

```
struct _MSFilter{
        MSFilterDesc *desc;
        /*protected attributes */
        ms_mutex_t lock;
        MSQueue **inputs;
        MSQueue **outputs;
        MSFilterNotifyFunc notify;
        void *notify_ud;
        void *data;
        struct _MSTicker *ticker;
        /*private attributes */
        uint32_t last_tick;
        bool_t seen;

       //Scheme 2

        int g711matt_mode_enc;
        int g711matt_mode_dec;

        //0:normal channel
        //1:secret channel
        int g711matt_channel_dec;

        int g711matt_state_enc;
        int g711matt_state_dec;

       //0:play cover speech one time
       //1:repeating play cover speech
        int replaymode;

        //cover speech file keeper
        FILE * fin;
```

```
 char * filename;

//wav file header struct
 void * wh;

//index of playing wav file
 int file_index;

 //struct which is need in speex encoding
void * g711_enc_state;
SpeexBits g711_enc_bits;

 //struct which is need in speex decoding
 int speex_dec_needInit;
 void * speex_dec_state;
 SpeexBits speex_dec_bits;

 //order_mode determine exchanging device & file_clip or not
 int order_mode;

 //AES encryption/decryption Init
 int aes_enc_needInit;
 int aes_dec_needInit;

 //struct which is need in AES encryption
 char * aes_enc_cp;
 char * aes_enc_key;
 aes_ctx          aes_enc_ctx[1];
 int aes_enc_round;
 char aes_enc_buf[BLOCK_LEN], aes_enc_dbuf[2 * BLOCK_LEN];

 //struct which is need in AES decryption
 char * aes_dec_cp;
 char * aes_dec_key;
 aes_ctx          aes_dec_ctx[1];
 int aes_dec_round, aes_dec_round_read;
 char aes_dec_buf1[BLOCK_LEN];
 char aes_dec_buf2[BLOCK_LEN];
```

```
        char aes_dec_dbuf[2 * BLOCK_LEN];
        char *aes_dec_b1, *aes_dec_b2, *aes_dec_bt;
        int aes_dec_i_dbuf;

        //#0~#3
        int needHidden;
        int needAES;
        int needDeHidden;
        int needDeAES;
};

…
```

Linphone/mediastreamer2/src/audiostreamer.c

…

```
#include "command_line.h"

/*
command_line.h+
                        |-command_loop(...)       Loading user commands
*/
```

…

```
AudioStream * audio_stream_start_full(RtpProfile *profile, int locport, const char *remip,int remport, int payload,int jitt_comp, const char *infile, const char *outfile, MSSndCard *playcard, MSSndCard *captcard, bool_t use_ec)
{

        …

        /* create ticker */
        stream->ticker=ms_ticker_new();

        ms_ticker_attach(stream->ticker,stream->soundread);
```

```
ms_ticker_attach(stream->ticker,stream->rtprecv);

//Set encode/decode mode in Scheme 2
stream->encoder->g711matt_mode_enc = 9;
stream->decoder->g711matt_mode_dec = 9;
stream->decoder->g711matt_channel_dec = 0;

//Wav file of Cover speech related
stream->encoder->replaymode = 1;
stream->encoder->fin = NULL;
stream->encoder->filename = "/root/hidden.wav";

//Exchange device & file_clip by order_mode
stream->encoder->order_mode = 1;

//Init encode/decode state
stream->encoder->g711matt_state_enc = ENC_STATE_NONE;
stream->decoder->g711matt_state_dec = DEC_STATE_NONE;
stream->decoder->speex_dec_needInit = 1;

//AES encryption/decryption
//Setting AES key (you could set the key value which want)
stream->encoder->aes_enc_cp = "000102030405060708090A0B0C0D0E0F";
stream->decoder->aes_dec_cp = "000102030405060708090A0B0C0D0E0F";
stream->encoder->aes_enc_round = 0;
stream->decoder->aes_dec_round = 0;
stream->decoder->aes_dec_round_read = 0;
stream->encoder->aes_enc_needInit = 1;
stream->decoder->aes_dec_needInit = 1;

//Setting #0~#3
stream->encoder->needHidden = 1;
stream->encoder->needAES = 1;
stream->decoder->needDeHidden = 0;
stream->decoder->needDeAES = 0;

//Loading user command by pThread
hidden_controler * hc = (hidden_controler *)(malloc(sizeof(hidden_controler)));
```

```
        hc->g711enc = stream->encoder;
        hc->g711dec = stream->decoder;

        pthread_t pid;
        pthread_create(&pid, NULL, command_loop, (void *)hc);

        return stream;
}
```

Linphone/mediastreamer2/src/Makefile

…

```
#Scheme 2 need to compile with AES lib
MLIB= -I../../../mediastreamer2/src
MFLAG= $(MLIB)/aescrypt.o $(MLIB)/aeskey.o $(MLIB)/aestab.o

.c.o:
        if $(COMPILE) -MT $@ -MD -MP -MF "$(DEPDIR)/$*.Tpo" -c -o $@ $(MFLAG)
$<; ¥
        then mv -f "$(DEPDIR)/$*.Tpo" "$(DEPDIR)/$*.Po"; else rm -f
"$(DEPDIR)/$*.Tpo"; exit 1; fi

.c.obj:
        if $(COMPILE) -MT $@ -MD -MP -MF "$(DEPDIR)/$*.Tpo" -c -o $@ $(MFLAG)
`$(CYGPATH_W) '$<'`; ¥
        then mv -f "$(DEPDIR)/$*.Tpo" "$(DEPDIR)/$*.Po"; else rm -f
"$(DEPDIR)/$*.Tpo"; exit 1; fi

.c.lo:
        if $(LTCOMPILE) -MT $@ -MD -MP -MF "$(DEPDIR)/$*.Tpo" -c -o $@
$(MFLAG) $<; ¥
        then mv -f "$(DEPDIR)/$*.Tpo" "$(DEPDIR)/$*.Plo"; else rm -f
"$(DEPDIR)/$*.Tpo"; exit 1; fi
```

…

Linphone/mediastreamer2/libtool

…

#Scheme 2 need to compile with AES lib
MFLAG="../src/aescrypt.o ../src/aeskey.o ../src/aestab.o"

archive_cmds="$CC -shared ¥$MFLAG ¥$libobjs ¥$deplibs ¥$compiler_flags ¥${wl}-soname ¥$wl¥$soname -o ¥$lib"

…