UNIVERSITI
TEKNOLOGI
PETRONAS

Implementation of a Symmetric Chaotic Encryption Scheme

by

Easwari Sivanandan
1621

Dissertation submitted in partial fulfilment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical and Electronics Engineering)

May 2004

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Implementation of a Symmetric Chaotic Encryption

by

Easwari Sivanandan

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,

[illegible]

(Associate Prof. Dr Varun Jeoti)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

May 2004

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

EASWARI SIVANANDAN

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Voice over Internet Protocol technology (VoIP) is progressing commendably, but packet loss, propagation delay, jitter, unreliable IP networks, and vulnerability to attacks by Internet hackers are among critical issues that have been identified. Voice privacy and security needs to focused upon and data encryption techniques are the answers in providing the security needed. However, traditional cryptosystems demand high computational complexity and high digital signal processors which in return increases the cost of implementation.

There is parallel growth in cryptographic techniques which originated an intense research activity and the search for new directions in cryptography such as chaotic encryption. Due to its deterministic nature and its sensitivity to initial conditions, chaos has a certain potential in creating a new way of securing information to be transmitted or stored.

There are two main objectives to this project. First is study the feasibility of the chaotic encryption scheme in providing a solution in to preserve data security while maintaining the voice quality for voice over Internet Protocol. Secondly, a new scheme based on a chaos system will be implemented for voice data. In order to achieve the second objective, a study had been carried out on other proposed schemes mainly the Hierarchical Data Security Protection (HDSP) for VoIP. This scheme performs two main operations which is the data-frame interleaving and intra-frame data encryption using bit swapping. Based on the HDSP scheme, the author suggests a new scheme using two level encryption techniques, based on chaos. In this scheme, the author uses the bit swapping technique as the second encryption-decryption level and enhances it with a first level encryption-decryption scheme using the two's compliment overflow nonlinearity encoder-decoder pair.

The implementation of this scheme is specified to do real time processing of voice data. It can also be used to read, encrypt and write a wave file. The entire system is implemented, tested and validated using MATLAB and Visual C++.

Due to the promising prospect of chaotic encryption in the field of cryptography, and the lack of implementation of this new encryption-decryption algorithm, this project focuses on introducing a new symmetric encryption-decryption scheme based on a chaos system for VoIP.

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND OF STUDY

The modern telecommunications networks, the Internet and mobile telephones have expanded the possibilities of user communications and information transmission to certain limits which were unimaginable not a very long time ago. With the great demand in digital signal transmission and the big losses from illegal data access, data security has become a critical and imperative issue. The increasing development of the broadband networks and services, alongside recent demand for privacy and paid services, has led to need for systems and algorithms to encrypt information [4]. Some of the important applications include encryption of video messages, voice over Internet protocol (VoIP), and data messages transmitted over telematic networks i.e. electronic banking and e-commerce.

Voice over Internet protocol network application meets the challenges of combining voice networks with packet networks by allowing both voice and signaling information to be transported over the packet network. In short, it means to transmit voice over the Internet. VoIP technology is progressing admirably, but certain drawbacks have been indicated such as packet loss, propagation delay, jitter, unreliable IP networks, and vulnerability to attacks by Internet hackers. Thus, it is essential to perform data encryption on voice transmitted over the Internet while preserving the quality of the voice from packet loss. There are researches on voice encryption using scrambling techniques, some of which uses discrete Fourier transform (DFT) to perform voice encryption and decryption in the frequency domain [1]. Although the performance has been good, but the computational power is large because of the forward and inverse transformations. It also increases cost as it requires a high performance digital processor to realize the transformations in real-time voice applications.

Encryption[1] technology is the practice of obscuring the privacy and security of the information by changing readable text to gibberish. The science of encryption itself is called cryptography. The significance of cryptography is evident as a method to secure communication and data authentication that offers confidentiality, data integrity, data and entity authentication. Encryption techniques are being rapidly developed, standardized and adopted such as RSA, DES, AES and etc. With the advancement in digital communication technology, the importance of cryptography in ensuring security and privacy for critical data transmission has been further increased with the need for secure communication and data authentication method. With the Internet, taking over as the preferred medium of communication, the demand for data encryption techniques to provide message security and authentication has been steadily increasing.

Hence, there is a parallel growing demand of cryptographic techniques which originated an intense research activity and the search of new directions in cryptography. As a result a rich variety of cryptosystems for end to end communications have been put forward, where robustness and privacy are equally diverse.

Chaos has attracted much attention in the field of cryptography for private and secure communications due to its deterministic nature and its sensitivity to initial values. Such properties mean that chaos has certain potential in creating a new way of securing information to be transmitted or stored. Methods based on chaos theory provide high level of security and are competitive to traditional encryption techniques because they are inexpensive to implement [4] and posses low computational complexity. Chaos and cryptography have common features, the most prominent being sensitive to variables and parameter changes. Shannon in his paper [C&C] wrote: “In a good mixing transformation…functions are complicated, involving all variables in a sensitive way. A small variation of any one variable changes the output considerably.” Chaos is the complex behavior of a simple system and it is very difficult to reproduce or decode, which makes it an effective means to secure communications.

[1] The process of changing the original data to gibberish or cipher text

1.2 LITERATURE REVIEW

Many efforts have been put into developing chaos-based cryptographic systems in recent years as emerging chaotic systems promised a new direction for innovation in cryptography since the development of symmetric and asymmetric cryptography. Many of these works are published, which shows a great deal of improvement and advancement over the years.

One of such work is the Hierarchical Data Security Protection scheme. This technique has been proposed in [1] as solution to the drawbacks VoIP has encountered. This scheme maintains both the voice quality which is degraded from the packet loss and provides high data security for the voice being transmitted. This scheme proposes two methods of data security i.e. inter-frame data interleaving and intra-frame data encryption. These operations will be discussed in detail in Chapter 2. This scheme as been used as the base of the author's research. Based on this work, another scheme has been implemented

The chaos based cryptosystem is new and has all the profound qualities for research. There are crucial similarities between cryptography and chaos. The equivalent of rounds, keys and diffusion in cryptographic algorithms in chaotic systems are iterations, parameters and initial change sensitivity, respectively. Using chaos in data encryption possesses features of low computational complexity, high security and no distortion. Compared to traditional encryption techniques, chaotic systems are competitive because they are inexpensive and easy to implement [5].

A foundation has thus been set to study the feasibility and applicability of an encryption-decryption scheme for voice data based on chaos systems. It is necessary to study the feasibility and applicability of the chaos encryption scheme with emphasis on the strength of the cipher. The strength of the system should be compared to current cryptosystems standards. The secrecy level should at least be equal if not greater than these standards.

1.3 PROBLEM STATEMENT

The introduction of chaotic systems into cryptography marks a very interesting area for research into innovating novel encryption schemes. In 1998, Baptista proposed Chaotic Encryption method which is said to be better than the traditional methods present today due to its randomness, stochastic, and mixed-ergodic characteristic.

The drawbacks of the Voice over Internet Protocol need to be addressed so that it can emerge from a specialized application to mainstream voice communications. The main drawbacks indicated are voice quality degradation due to packet loss and vulnerability to attacks by hackers. The current schemes used to do real-time encryption require high computational power and high performance digital signal processors which increase the cost [1]. Hence, the author is carrying out a research to find new schemes which can preserve high data security. Since chaos is a new direction in the field of cryptography with promising results such as low computational complexity, high security and no distortion, this project proposes an implementation of a new scheme based on chaos which can overcome the drawbacks of the Voice over Internet Protocol focusing on data security.

1.4 OBJECTIVE AND SCOPE OF STUDY

i. To study and develop a new security system based on the HDSP scheme. The HDSP scheme consist data interleaving for voice quality maintenance and intra-frame data security based on pixel permutation using chaotic bit strings for VOIP network.

ii. To study and propose a method on how chaotic encryption can be used to secure voice communications over a VoIP network.

iii. To research, study and propose a chaotic encryption algorithm using other chaotic maps i.e. logistic map

iv. Convert the chaotic encryption scheme based on two's compliment overflow nonlinearity encoder-decoder developed in Simulink to C.

v. To enhance the chaotic encryption scheme in (ii) by proposing an additional level of encryption using permutation techniques in C.

vi. To encrypt voice data sent in real-time.

The development of the proposed symmetric chaotic encryption scheme will be developed using MATLAB, Simulink and C. The end system will be a standalone C application which can run in terminal connected to a network. The scope of study will focused on data encryption techniques applicable to VoIP and will not cover techniques to reduce packet loss or maintain voice quality in a VoIP system.

1.5 THESIS ORGANIZATION

The chapters are organized to provide a flow to the report and also to aid the reader in understanding the project in a more systematic and ordered manner.

Chapter 1 highlights the overview or background on the project and literature review to justify the basis of the project followed by the problem statement. The objectives and scope of work are also outlined in this chapter.

Chapter 2 discusses the theories involved in the project and provides a base for the reader to understand the operations and implementations involved in this project. This chapter helps the reader obtain supporting information and the theories used in this project.

Chapter 3 then discusses the methodology of implementation and algorithms used in realizing the final product.

Chapter 4 further elaborates the results and observations obtained and makes a comparison to the theories reviewed in Chapter 2. All findings and discussions are highlighted here.

Finally, chapter 5 concludes the project report by cross-checking the relevancy to the objectives, suggested future work and recommendations.

CHAPTER 2

THEORY

2.1 DATA SECURITY IN NETWORKED ENVIRONMENT

In the recent years, there has been a tremendous upsurge in information and data transfers over the telephone and computer networks. Connectivity has become a buzz word in industries and the increasing sophistication of network software and controllers is allowing the average computer users' access to information never thought possible a decade ago. This information ranges from very simple electronic mail to highly complex medical imaging.

In any of these data transfers, the security of the information being transferred is a pressing concern. However with this increase to global databases, the need to secure/protect information from browsers, whether during transmission, reception, and storage risen dramatically. The increasing development of broadband networks and services, alongside the recent demand privacy, has led to the need for systems and algorithms to encrypt information. The most vital applications include the encryption of video messages, voice over Internet Protocol (VoIP) and data messages transmitted over telematic networks i.e. electronic banking and commerce. Currently the solution to the security concern is to use expensive and inefficient private networks and leased lines.

Widely used encryption and keying schemes based upon Data Encryption Standard (DES), a secret key cryptography, and Rivest-Shamir-Adleman (RSA) a public key cryptography. Initial investigations [19] show that a HYBRID cryptographic approach where public key is used for authentication or key management and the symmetric key approach for encryption would be most appropriate for This project discusses the latter, symmetric key encryption based on a chaos system for security that will allow voice data to be transmitted over the IP network so that only the intended receiver can decipher.

2.2 VOICE OVER INTERNET PROTOCOL (VoIP)

Voice over Internet protocol network application meets the challenges of combining voice networks with packet networks by allowing both voice and signaling information to be transported over the packet network. VoIP wants to provide the efficiency of a packet switched network while rivaling the voice quality of a circuit switching network. VoIP technology is progressing admirably, but certain drawbacks have been indicated such as intolerant of lengthy delays, out of order packets, packet loss, propagation delay, jitter, unreliable IP networks, and vulnerability to attacks by Internet hackers. All these problems gravely degrade the quality of voice transmitted to the recipient.

The overall technology requirements of an Internet Protocol (IP) solution can be split into four categories: signaling, encoding, transport and gateway control. The purpose of the signaling control is to create and manage connections between endpoints, as well as the calls themselves. Next, when the conversation commences, the analog signal produced by the human voice, needs to be encoded for transmission across an IP network. The IP network itself must ensure that the real-time conversation is transported across the available media in a manner that produces acceptable voice quality.

It is essential to perform data encryption on voice transmitted over the Internet while preserving the quality of the voice from packet loss. There are researches on voice encryption using scrambling techniques, some of which uses discrete Fourier transform (DFT) to perform voice encryption and decryption in the frequency domain [1]. Although the performance has been good, but the computational power is large because of the forward and inverse transformations. It also increases cost as it requires a high performance digital processor to realize the transformations in real-time voice applications.

2.2.1 Voice Quality

Many factors determine the voice quality, including the choice of codec, packet loss, echo control, delay, delay variation and the design of the network. Packet loss causes voice clippings and skips. Some codec algorithms can correct for some lost voice packets. Techniques such as interleaving and error correction using CRC can be used to maintain the voice quality. The concept of data interleaving is used to scramble the voice packets in different frames to avoid continuous voice corruption due to packet loss.

TCP is the internet protocol that suites main transport layer protocol. It also provides addressing (with service addresses) services at the network layer. TCP provides reliable full-duplex, connection-oriented transport services to upper layer protocols. TCP works in conjunction with IP to move packets through the internetworking. TCP assigns a connection ID (port) to each virtual circuit. It also provides message fragmentation and reassembly using sequence numbering. Error checking is enhanced through the use of TCP acknowledgements.

UDP is a connectionless protocol that works at the transport layer. UDP also transports datagram but does not acknowledge their receipt. UDP also uses port address to achieve datagram delivery, but this port address is simply a pointer to a process, not a connection identifier, as it is with TCP. Mainly, voice packet transmission uses UDP.

To make VoIP successful in real time networking interactive applications, guaranteed QOS or quality of service networks are inevitable for minimizing the delay caused by the packet network transmission. It is reasonable to assume that the packet network transmission delay of a guaranteed QOS is ~25ms [1].

The proposed system in this project however does not cover the framework of packet loss and only focuses on data security.

2.3 ENCRYPTION-DECRYPTION

In this project we deal with the theory of encryption and decryption, existing encryption standards, and mainly chaotic dynamical systems and their use in solving practical problems like implementing an efficient cryptographic system.

Encryption standards are evaluated for their suitability according to three main criteria:

- Security
- Cost
- Algorithm and implementation characteristics

"Security" of the proposed algorithm was absolutely essential and any algorithm found not to be secured would not be considered further. "Cost" refers to the computational efficiency i.e. speed and memory requirements of various types of implementations including software, hardware and smart cards. "Algorithm and implementation" characteristics include flexibility and algorithm simplicity among other factors.

From the author's understanding, on "what makes a good encryption algorithm", though not complete, is outlined as below:

- Iterations required per sample
- Speed of encryption and decryptions
- Error propagation (when you iterate something, i.e. current output depends on the previous output etc, they tend to produce error propagation)

Thus, these are the criteria to look into when comparing two encryption schemes.

A need to replace DES had arisen and the replacement was to be called AES or Advanced Encryption Standard. Rijndael was selected to be the Advanced Encryption Standard based on its combination of security, performance, efficiency, implement ability, and flexibility.

2.3.1 AES

AES algorithm is a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths; however they are not adopted in this standard.

AES is a key-iterated block cipher; it consists of the repeated application of a round transformation on the state. The number of rounds denoted by Nr depends on the block length and key length. Table 2.1 lists the value of Nr as a function of Nk and Nb. For AES, Nb is fixed to a value of 4. Nr=10 for key length 128 bits, Nr=12 for key length 192 bits and Nr=14 for key length 256 bits. For AES, the value of Nb is set to 4 since the block length is 128 bits.

Table 2.1. Number of rounds (Nr) as a function of Nb (Nb = block length/32) and Nk (Nk = key length/32)

N_k	N_b				
	4	5	6	7	8
4	10	11	12	13	14
5	11	11	12	13	14
6	12	12	12	13	14
7	13	13	13	13	14
8	14	14	14	14	14

The input and output of AES is considered to be one-dimensional arrays of 8-bit bytes consisting of 128 bits (digits with values of 1 or 0). These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. For encryption the input is a plaintext block and a key and the output is cipher text block. For decryption, the input is a cipher text block and a key and the output is the plaintext block. The key for the AES algorithm is a sequence of 128, 192 or 256 bits. Other input, output and key lengths are not permitted by this standard. The bits within such sequences will be numbered starting at zero and ending at one less than the sequence length (block length or key length). The number i attached to a bit is known as its index and will be in one of the ranges $0 \leq i < 128$, $0 \leq i < 192$ or $0 \leq i < 256$ depending on the block length and key length (specified above).

The round transformation is denoted by Round and is a sequence of four transformation called steps. The structure of the AES round transformation requires that all steps be invertible and consist of simple components rather than complex ones.

According to [21] and [22], a high-level description of the AES algorithm can be summarized as below:

I. Given a plaintext, x, initialize State to be x and perform an operation ADD-ROUNDKEY[3], which x-ors the RoundKey with State.
II. For each of the first Nr-1 rounds, perform a substitution operation called SubBytes on State using an S-Box; perform a permutation ShiftRows on State; perform an operation MixColumns on State; and perform AddRoundKey.
III. Perform SubBytes; perform ShiftRows; and perform AddRoundKey.
IV. Define the cipher text y to be the State.

These processes are required to perform an encryption operation in AES. To decrypt, it is necessary to perform all operations in the reverse order, and use the key schedule in the reverse order. Further, the operations ShiftRows, SubBytes, MixColumns, must be replaced by their inverse operation including the operation AddRoundKey. It is also possible to construct an equivalent inverse cipher which performs AES decryption by doing a sequence of inverse operations in the same order as it is done for AES encryption. It is suggested in [21] that this can lead to implementation efficiencies.

[3] Round keys are values derived from cipher key using the Key Expansion routine; they are applied to the State in the cipher and inverse cipher

2.3.2 Analysis of AES

The design criteria for AES were looked upon during this analysis on why the 4 steps are performed as such in each round. The table 2.2 below summarizes the operations, design criteria and its importance.

Steps	Design Criteria and importance
1. SubBytes	1. **Non-linearity** a) **Correlation.** The maximum input-output correlation amplitude must be as small as possible. b) **Difference propagation probability.** The maximum difference propagation probability must be as small as possible. 2. **Algebraic Complexity.** The algebraic expression of the S-box[4] in GF (2^8) must be complex.
3. MixColumns	1. **Dimensions.** The transformation is bricklayer transformation operating on 4-byte columns. 2. **Linearity.** The transformation is linear over GF(2) 3. **Diffusion.** The transformation has to have relevant diffusion power. 4. **Performance on 8-bit processors.** The performance of the transformation has to be high.
4.Key schedule	Key Expansion 1. **Efficiency** 2. **Performance** 3. **Symmetry Elimination** 4. **Diffusion** 5. **Non-linearity**

[4] Non-linear substitution table used in several byte substitution transformations and in the Key Expansion routine to perform a one for-one substitution of a byte value.

2.3.3 Cryptanalysis

The current state-of-the-art in cryptanalysis indicates that the resistance of iterative block ciphers against cryptanalytic attacks increases with the number of rounds.

According to [21] and [23] the AES is secure against all known attacks. This is one of the main reasons AES is chosen to be compared against chaos encryption. Various aspects of it design incorporate features that help provide security against specific attacks. For example, the used of the finite field inversion operation in the construction of the S-Box yields linear approximation and difference distribution tables in which entries are close to uniform. This is the main factor which helps provide security against differential and linear attacks. As well, the linear transformation, MixColumns makes it impossible tot find differential and linear attacks that involve 'few' active S-boxes. There are apparently *no known attacks* on AES that are faster than exhaustive search. The 'best' attacks on AES apply to variants of the cipher, which the number of rounds is reduces and are not effective for 10-round AES.

2.4 CHAOS

In signal theory, chaos is classified as one of the non-periodic signal systems which are characterized by a continuous noise-like broad power spectrum. Chaotic signals are naturally broadband and are noise-like. Chaotic systems are characterized by sensitive dependence on initial conditions where a small perturbation eventually causes a large change in the state of the system itself. The practical long-term prediction of these signals is not possible due to the high dependence on initial conditions. Chaotic systems also decorrelate rapidly with themselves. The autocorrelation function of a chaotic signal has a large peak at zero and decays rapidly. They also posses deterministic structure that makes it possible to generate noise like chaotic signals in a theoretically reproducible manner. Chaos based algorithms offers a high degree of randomness, stochastic and mixed-ergodic properties.

2.4.1 Chaos and Cryptography

Chaos describes a system that is sensitive to initial conditions, generates apparently random behavior but at the same is completely deterministic. These properties have much potential for any application in cryptography as it is hard to make long term predictions on chaotic systems.

Firstly, chaotic functions are sensitive to initial conditions thus any slight change in the initial value will yield a drastically different ciphertext. This indicates that the system will be strong against any brute force attacks as the number of possible keys is enormous given that the precision of initial value is high.

Secondly, being deterministic means that the same mapping function and initial value will yield the same set of values. Conventional random number generators cannot regenerate random numbers where as chaos allows us to repeat the same string as long as the same mapping functions and initial value is used. The apparent randomness of the system makes attacks such the 'codebook' attack impossible.

Claude Shannon had pointed out two properties which make a good cryptographic system hinder statistical analysis. Diffusion means that changing a character of the plaintext would affect several characters in ciphertext and likewise changing a character of ciphertext would affect several characters in the plaintext. Confusion is a technique which corresponds to those parts of a cipher mechanism which change the association between the input values and output values. This is generally accomplished by substituting every fundamental block for another one according the rules dictated by the algorithm. This technique confuses the output by modifying the data itself.

Analyzing an example of a cipher using chaos, given a set of plaintext which is to be encrypted in integer form, a mapping function must be defined i.e. logistic map. Next, is to define the initial value, n, of the mapping function. Thus, the first character in the ciphertext will use n as the initial value and first number in the plaintext as the number of iterations to yield the ciphertext character. The next ciphertext character will be obtained using the previous ciphertext character as the initial value, while iterating the number of times indicated by the present plaintext character. Hence, any change in plaintext would affect the remaining part of the ciphertext. This means that diffusion is introduced in the system. Secondly, the characters in the ciphertext will not only depend on the key but the plaintext itself. Thus there are elements of confusion as well in the system. As long as the mapping function is not disclosed, the system can withstand statistical attacks.

2.4.2 Chaotic Encryption and its Algorithm

Chaotic map is a chaotic system with the property to evolve identically if the initial conditions are the same. The chaotic map is the main part of the encryption system. The more complex the map, the more difficult to force it. One way to enhance the complexity of the maps for encryption is by mixing them. The real important part is to be able to reproduce the maps the way it has been mixed. It will be more difficult now to calculate all the possible situations.

The key is the only piece of information unknown to the attacker. The key could be public knowledge and this fact should not compromise the system. One way of solving it is by means of chaotic dynamical system. There are several methods by which the plaintext can be combined with a chaotic sequence to produce seemly cipher text, one that appears random and disordered by permutation or invertible substitution. The digits or the letters of the cipher text can be calculated from the original and the key as follows:

$$ci = Ck\,(tk) = ti + k\ (mod\ m) \qquad (1)$$

where *ci* is the cipher text's i-th digit, *ti* the i-th digit of the original text and *k* the actual value of the chaotic sequence. *m* is assumed to be the alphabet size. Decoding is simply performed by the inverse operation of (1).

$$ti = ci - k \ (mod\ m) \qquad (2)$$

This is possible for the recipient to decode because he knows the key and can reconstruct the chaotic sequence appearing in (2). On the other hand, it will be difficult for the hacker because he lacks the information of the original conditions of the system that led to the production of k. Thus, the sender, recipient and hacker are aware of the chaotic system being used but only the sender and recipient share the knowledge of the exact initial condition of the system. This constitutes the key. The hacker may try to guess the conditions but his chance of success is inversely proportional to ***2n*** where ***n*** is the number of effective bits used to describe the initial conditions. Even if he is able to come up with a close guess, it would not help because of the chaotic system's sensitivity to initial conditions would multiply any original discrepancy in a few iterations. It is possible to generate efficient ciphers using for example a quadratic map exhibiting fully developed chaos. Variations on this theme are easy to come by, instead of resorting to substitution scheme, one might use a chaotic system to generate a permutation matrix and use this to encode a message.

The key advantages of the chaos encryption are resistance to traditional form of attacks because it is not written like standard algorithms, ease in increasing the variety of algorithms, difficulty in detecting spectral peaks and suitability of implementation in analogue systems [10]. It is a simple method of encryption and can be achieved by iteration with a high level of security. There is no short cut and the security depends on the key.

But as with every system, there is always a flip side, and chaos cryptography is no exception. Due to difficulty in cryptanalysis the security cannot be readily quantified [10]. The system is also said to be insecure to be encrypting long messages. This weakness is attributed to the fact that chaos mappings can go in an orbit or repeat its pattern for various initial conditions. It is also slow in speed and requires a high precision of floating point calculation and large number of iteration.

2.4.3 Chaotic encryption using logistic map

The logistic map is given by the equation:

$$x(n+1)=r*x(n)*(1-x(n)) \qquad (3)$$

where the value of the bifurcation parameter, r should be selected as the values that can result in chaos and $0 < x(0) < 1$. The successive states from the logistic map is derived using equation in (3) and the preceding 16 or 24 bits below the decimal point of the binary representation of x(n) where n= 1, 2, ... are extracted to constitute the chaotic bit string sequence b(0), b(1), b(2),.....In the chaotic systems [1], there are perfect statistical characteristics such as (i) sensitive dependence on initial condition and (ii) there exist trajectories that are dense, bounded but neither periodic nor quasi-periodic in the state space. Since the one dimensional logistic map has the property of easy realization, it is adopted to generate the unpredictable sequence b(.).The logistic equation is as the equation in (1) and is parabolic like the quadratic mapping with f (0) = f (1) = 0. The value of u is selected from a range of values which can cause chaos based on the bifurcation diagram and $0 <x(0) <1$. The value of x (n) is the nth value in the sequence and then folds it back by (1-x (n)). The author carried out research on how to determine the value of u such that it would create chaos. According to [5] and [14], the parameter u varies in the interval [0, 4] so that u will be values that can generate chaos and x (0) will stay in the range of 0 to 1. This can be observed from the bifurcation diagram in Figure 2.1.

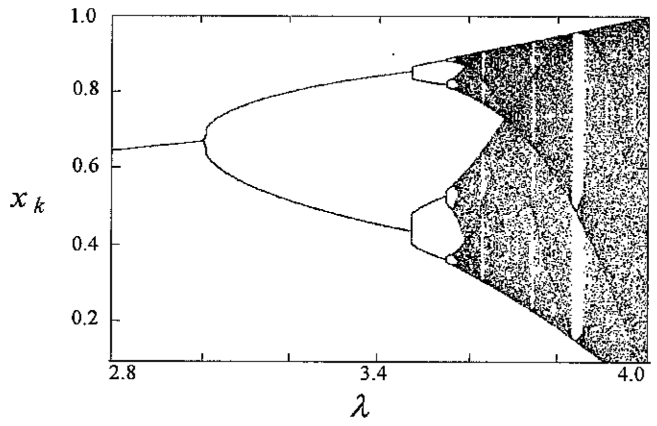


Figure 2.1 *Bifurcation Diagram*

The bifurcation parameter takes the range 0<u<4 and determines the chaotic attractor. In the range 0<u<1, x tends to zero while 1<u<4 leads to a point attractor. At u≈3 the first bifurcation occurs, the curve bifurcates again at u≈3.45 and the period doubling continues at ever closer u values to chaos. At u=4, complex chaotic behavior can be observed.

2.4 THE HIERARCHICAL DATA SECURITY PROTECTION (HDSP)

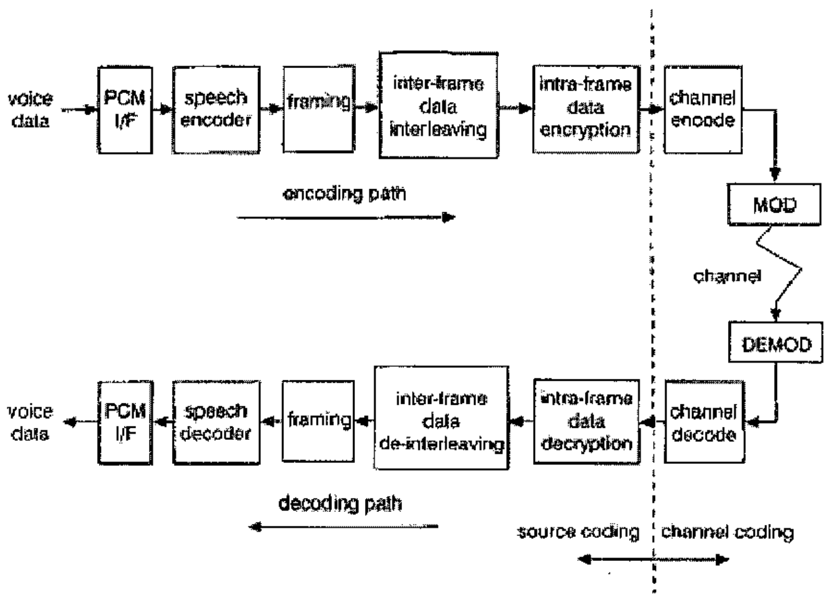


Figure 2.2 *Block diagram of proposed HDSP scheme applied in VoIP*

This technique has been proposed in [1] as solution to the drawbacks VoIP has encountered. This scheme maintains both the voice quality which is degraded from the packet loss and provides high data security for the voice being transmitted. This scheme proposes two methods of data security i.e. inter-frame data interleaving and intra-frame data encryption. These operations are controlled by chaotic bit-string generated from the chaotic bit-string generator based on the 1-D logistic map. The inter-frame data interleaving, permutation is adopted to perform frame re-ordering. Pixel value transformation including bit swapping and XOR or XNOR operations are adapted to translated the original pixel value to an encrypted one. This encryption algorithm falls in the combination form which performs both position permutation and value transformation. The value transformation has low computational complexity and low hardware cost. This combination normally exhibits the potential of high security. The block diagram in Figure 2.2 below illustrates the proposed HDSP scheme to be applied in VoIP. The main focus is the framing operation, inter-frame data interleaving and intra-frame data encryption. The framing operation would include breaking up the voice input to a defined number of packets. The chaotic bit string generator is based on the chaotic system utilized (chaotic maps). To generate the binary sequence $b(0)$, $b(1)$, $b(2)$,... from $x(0)$, $x(1)$, $x(2)$,..., the 8 bits before the decimal 16 point of the binary representation of $x(n)$ for n = 0, 1, 2,... are extracted to constitute the sequence. That is,

$$0.b(16n+0)b(16n+1)\ldots\ldots b(16n+14)b(16n+15) \qquad (4)$$

is the binary representation of x(n) for n = 0, 1, 2,......n.

2.5 Proposed Two-Level Security System for VoIP

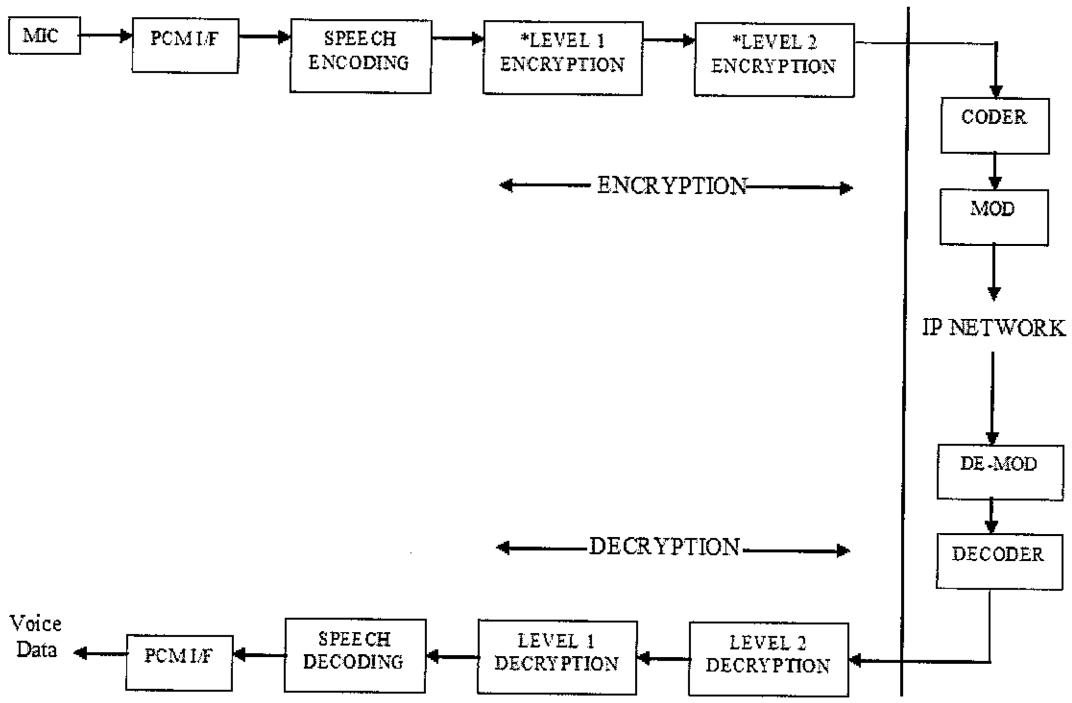


Fig. 2.3 *Block diagram of proposed security scheme for VoIP*

Based on the scheme in Figure 2.2, a new security scheme for VoIP as depicted by the block diagram above in Figure 2.3. The proposed level one encryption which is two's compliment overflow nonlinearity encoder-decoder and level two encryption which is the bit swapping are inserted in the traditional VoIP data flow between the speech CODEC and the channel CODEC. These operations are controlled by a chaotic bit string generator based on the chaotic system of a one-dimensional logistic map. The implementation of the encryption-decryption schemes will be discussed in Chapter 3 of Methodology and Project Work while the system will be formally discussed in Chapter 4 of Results and Discussion.

CHAPTER 3

METHODOLOGY and PROJECT WORK

3.1 Procedures and Methodology

The project work is divided into two parts of to ensure that the time frame of two semesters is fully utilized and organized well. For the first semester, the objectived outlined was to get acquainted with the theory of cryptography, focusing specifically on the Advanced Encryption Standard and Chaos Encryption and comparing results of both encryption schemes.

The objectives of the second semester has been clearly outlined in Chapter 1. Firsly, a thorough and intense research is carried out to gather suffiecient information and research papers pertaining chaos, chaos in cryptography, streaming, voice over Internet Protocol and its security issues. The research is done based on books, IEEE publications, the Internet, and also certain individuals inclined to this area of research. The information being gathered is important in understanding the role of chaos in cryptology, the concept of streaming to be applied in a cryptosystem to provide real-time encryption, the security issues in voice over Internet Protocol and encryption schemes to overcome it.

Secondly, the focus is to finalize on the chaotic encryption scheme implementation based on research carried out. Materials relevant to voice over Internet Protocol and its security issues have been researched upon and the author has decided to further explore the scheme proposed in [HDSP]. This scheme is discussed in Chapter 2 and is illustrated in Figure 2.2. The author will focus on implementing the framing block, and intra-frame data encryption as illustrated in Figure 3.1.

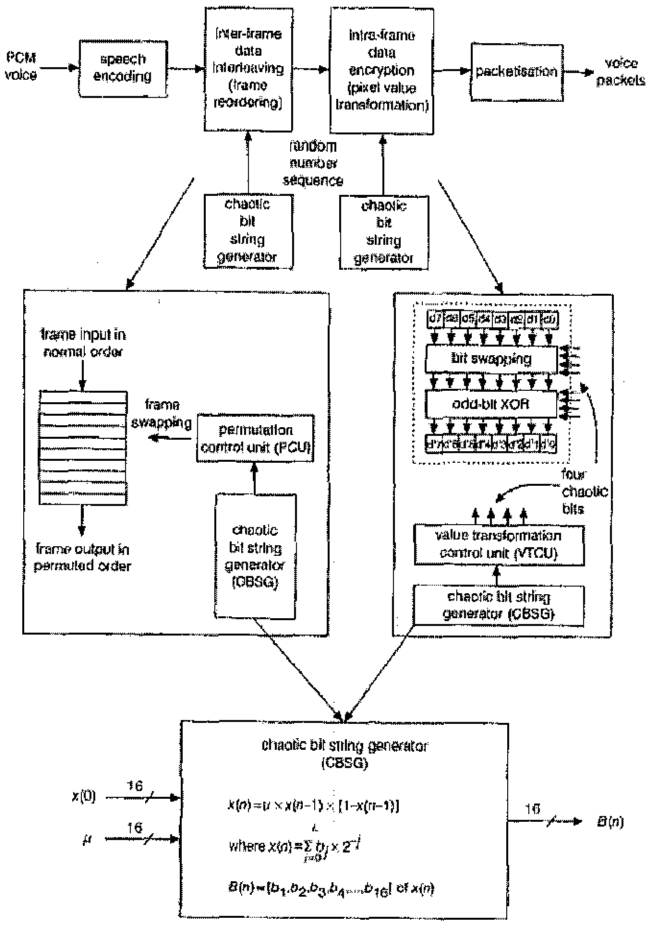

PCM voice
speech encoding
inter-frame data interleaving (frame reordering)
intra-frame data encryption (pixel value transformation)
packetisation
voice packets
random number sequence
chaotic bit string generator
chaotic bit string generator
frame input in normal order
frame swapping
permutation control unit (PCU)
chaotic bit string generator (CBSG)
frame output in permuted order
bit swapping
odd-bit XOR
four chaotic bits
value transformation control unit (VTCU)
chaotic bit string generator (CBSG)
chaotic bit string generator (CBSG)
$x(n)=u\times x(n-1)\times[1-x(n-1)]$
where $x(n)=\sum_{j=0}^{L} b_j\times 2^{-j}$
$B(n)=[b_1,b_2,b_3,b_4,\ldots,b_{16}]$ of $x(n)$
$x(0)$
μ
16
$B(n)$

Figure 3.1 *Operations of inter-frame data interleaving and intra-frame data encryption*

3.2 The implementation of chaotic bit string generator

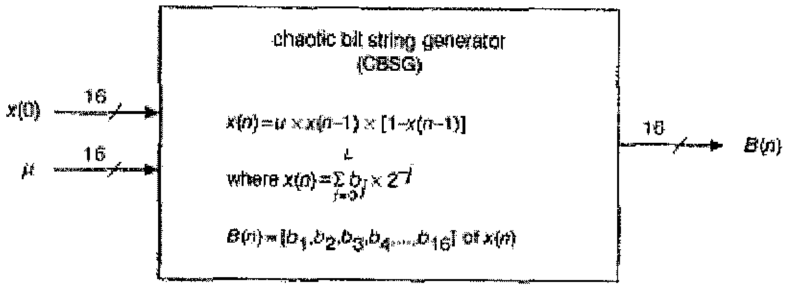

chaotic bit string generator (CBSG)
$x(n)=u\times x(n-1)\times[1-x(n-1)]$
where $x(n)=\sum_{j=0}^{L} b_j\times 2^{-j}$
$B(n)=[b_1,b_2,b_3,b_4,\ldots,b_{16}]$ of $x(n)$
$x(0)$
μ
16
16
16
$B(n)$

Figure 3.2 *The chaotic bit string generator*

In the proposed HDSP scheme there are three main operations i.e. framing, inter-frame data interleaving and intra-frame data encryption as illustrated in Figure 3.1. The operating intra-frame data encryption is controlled by the chaotic bit string generated from the chaotic bit string generator based on a one-dimensional logistic map as illustrated in Figure 3.2.

Determine the bifurcation parameter, r and initial point x(0) of the 1-D logistic map $f_r(x)$ *= rx(1-x) where r should be selected from a range of values which would result in chaos and 0 < x(0) <1. Evolve each successive state from the map by x(n+1)=rx(n)(1-x(n)) and the preceeding 16 bits below the decimal point of the binary representation of x(n) where n=1, 2, ... are extracted to constitute the chaotic bit-string sequence b(0), b(1), b(2) and so on.*

The chaotic bit string is generated by extracting 16 bits from each evolution state of the 1-D logistic map. This bit string is important as it is manipulated in the swap bit function. The successive states are evolved from the logistic map and the preceding 16 bits below the decimal point for the binary representation of x(n) are extracted to constitute the chaotic bit string sequence b(0), b(1), b(2), b(3).......b(16). To generate the chaotic binary sequence, the 16 bits before the decimal point of the binary representation of x(n) where n=0,1,2,..are extracted to constitute the sequence. For example for x (0), the 16 bit-binary representation

$$0.b\ (16n+0)b\ (16n+1)b\ (16n+2).....b\ (16n+7)$$

where of n=0, which returns b (0), b (1).......b (15). The chaotic bits would be passed to an array with N number of elements which is equal to the number of iterations and used in the bit swapping function.

3.3 Bit swapping algorithm

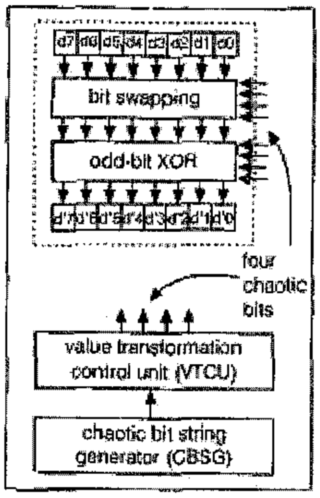

d7 d6 d5 d4 d3 d2 d1 d0
bit swapping
odd-bit XOR
d'7 d'6 d'5 d'4 d'3 d'2 d'1 d'0
four chaotic bits
value transformation control unit (VTCU)
chaotic bit string generator (CBSG)

Figure 3.3 *Intra-frame data encryption*

With reference to Figure 3.3, for intra frame encryption, the pixel value transformation which includes two operations of bit swapping and XOR operations based on the chaotic bit string b(0),b(1).....b(n) which transforms the original pixel value to an encrypted one. The algorithm for the bit swapping is discussed as follows. Each sample is represented by one byte which is 8 bits. The swapping will take place locally within the sample. The pairs to be swapped is predetermined as bit zero with bit 4 (0, 4), bit one and five (1, 5), bit two and six (2, 6) and bit three and seven (3, 7). Each pair will be assigned to one chaotic bit. If the bit is 1, then it will be swapped. If the bit is 0, the pair will remain. Then, the 1st, 3rd, 5th and 7th bit will be XORed to the four chaotic bits. This data encryption approach has the features of low computational complexity. The algorithm is as below:

g' is the result of the data encryption Definition 2: The operation $SwapBit_w(dr, ds)$ is defined as to swap bit dr and bit ds if w is equal to 1 or to preserve their original values if w is equal to 0.

***Step 1**: Determine the bifurcation parameter, r and initial point x(0) of the 1-D logistic map $f_r(x) = rx(1-x)$ where r should be selected from a range of values which would result in chaos and $0 < x(0) < 1$. Evolve each successive state from* *the map by using equation (3) and the preceeding 16 bits below the decimal point of the binary representation of x(n) where n=1, 2, ... are extracted to constitute the chaotic bit-string sequence b(0), b(1), b(2) and so on.*

```
Step 2: For ctr=0 to 4  where ctr equals 4 incoming samples at one time
            For bit= 0 to 3
                $SwapBit_{b(4\,xctr\,+bitl)}(d_{bit}, d_{bit+4})$
            End
            For odd_bit = 1 to 7 Step 2
                $d_{odd\_bit'} = d_{odd\_bit'}$ XOR b(4 x ctr + odd_bit)
            End
        End
```

***Step 3**: The encryption result g' is obtained and the algorithm terminated."*

The interframe data interleaving will be replaced by another encryption-decryption technique as illustrated in the Figure 3.4.

3.4 TWO's COMPLIMENT OVERFLOW NONLINEARITY

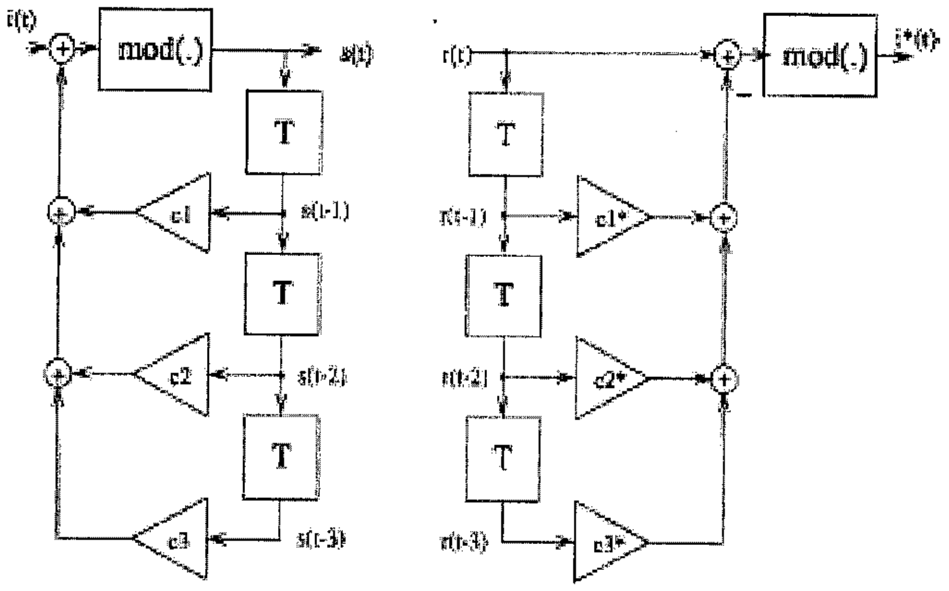


Figure 3.4 *Encoder-decoder using two's compliment overflow nonlinearity*

The encoder-decoder structure with two's compliment overflow non-linearity [2], special parameter sets and fixed-point, floating point or continuous value realization is used for coding of speech for another level of encryption.

Basically, this system is a generalized implementation of a simplified delay-based system. With reference to Figure 3.4, the encryptor portion of the system consists of all the blocks to the left of the transmission line, whereas the decryptor is located to the right.

It should be obvious that each key-pair (Gain 1 and Gain 4, Gain 2 and Gain5, Gain 3 and Gain 6) should have the same key value to ensure that the decryption process produces the correct recovered plaintext from a given cipher text.

The gains in the feedback loop of the system act as the encryption keys. The value for each key is obtained from a chaotic sequence generated using the logistic map. The mod function is recognized as the two's complement non linearity and mathematically defined as:

$$\mathrm{mod}(x) = x - 2.\lfloor (x+1)/2 \rfloor \tag{1}$$

where x, in this case, is $i(k) + \kappa(k)$, and $\lfloor . \rfloor$ is a flooring operation.

A simple generalization of delay based system implementation would then be:

$$s(k) = \mathrm{mod}(i(k) + \sum_{j=1}^{n} c_j s(k-j)) \tag{4}$$

and the decoded signal would be:

$$i^*(k) = \mathrm{mod}(r(k) - \sum_{j=1}^{n} c_j^* r(k-j)). \tag{5}$$

where c_j is the key and identical keys in need to be used in both the encoder and decoder. The key values are achieved by using the logistic map. But instead of converting the iterated values from the map to bits of ones and zeroes as done prior to bit swapping, the iterated values of x(n) itself will be used as the key values. This is because the key values need to be larger than one and never zero.

The block diagram in Figure 3.4 had been implemented in MATLAB Simulink. A C-standalone was created and was able to execute outside of MATLAB. The author reviewed the C-codes that were generated and it was completely using Simulink (MATLAB) API's and libraries linked into the program. Even audio capture (from microphone) is done via API's in Simulink and MATLAB. The makefile also seems to indicate the usage of Simulink's own compiler called LCC to compile the program.

From the findings, the author has discovered that Simulink has the libraries not only needed to do the mathematical equations but also communicate with the operating system to interface with the microphone. Unfortunately, the author could not discover a way of interfacing Matlab / Simulink with network as well, so that the whole system in Figure 4.1 can be modelled in MATLAB and from there create a standalone. However, assuming the worst case scenario that Matlab simulink cannot do networking or that the overhead interfacing Matlab / Simulink with network as well, so that the whole system in Figure 4.1 can be modelled in MATLAB and from there create a standalone. However, assuming the worst case scenario that Matlab simulink cannot do networking or that the overhead of the Matlab Simulink libraries cannot cope with "real-time-ness", then author continued to simulate the system using Visual C++.

These definitions and steps are implemented in MATLAB using m-files for initial development. The encryption algorithm was then enhanced and further optimized using C-coding. The end system is implemented in C and the results that was obtained from the encryption and decryption for real-time voice data is analyzed using MATLAB. Please refer to Appendix D to view the C-codes for the proposed two-level encryption scheme based on chaos. MATLAB provides tools such as spectograms, histograms and power density spectrum plots which enables one to analyze the input signal, encrypted signal and the output or decrypted signal. Comparisons and conclusions from the analysis can be drawn using the tools provided.

3.5 Flow Chart

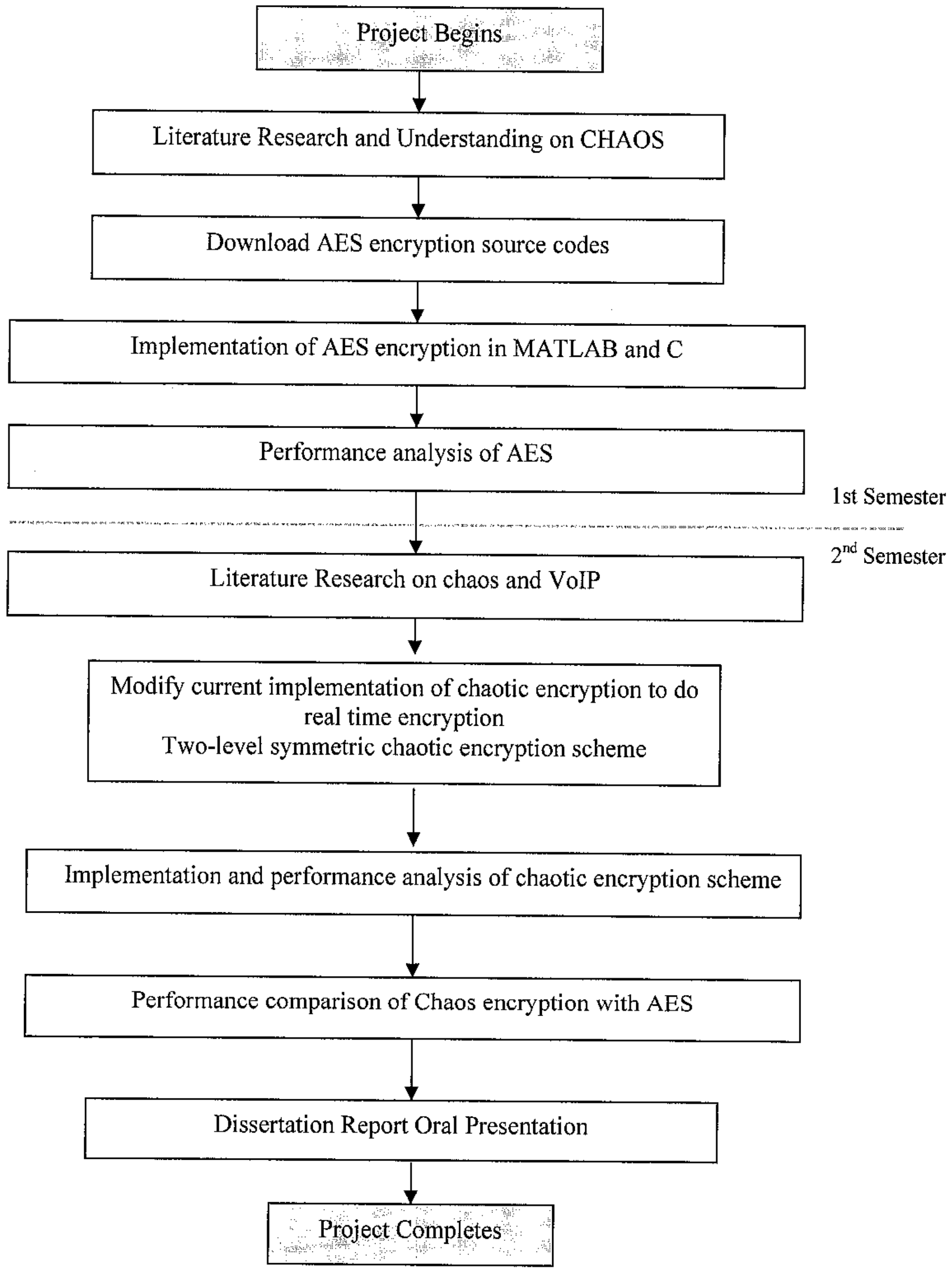

Project Begins
Literature Research and Understanding on CHAOS
Download AES encryption source codes
Implementation of AES encryption in MATLAB and C
Performance analysis of AES
1st Semester
2nd Semester
Literature Research on chaos and VoIP
Modify current implementation of chaotic encryption to do real time encryption
Two-level symmetric chaotic encryption scheme
Implementation and performance analysis of chaotic encryption scheme
Performance comparison of Chaos encryption with AES
Dissertation Report Oral Presentation
Project Completes

Figure 3.5 *Flow Chart of project work across two semesters*

3.6 Tools/Software

The software' used are MATLAB and Visual C++. The chaotic encryption scheme will be converted to C-standalone executable program. Hence, this encryption program can be installed in any computer as a standalone system.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 PROPOSED SECURITY SCHEME FOR VOICE OVER INTERNET PROTOCOL (VOIP)

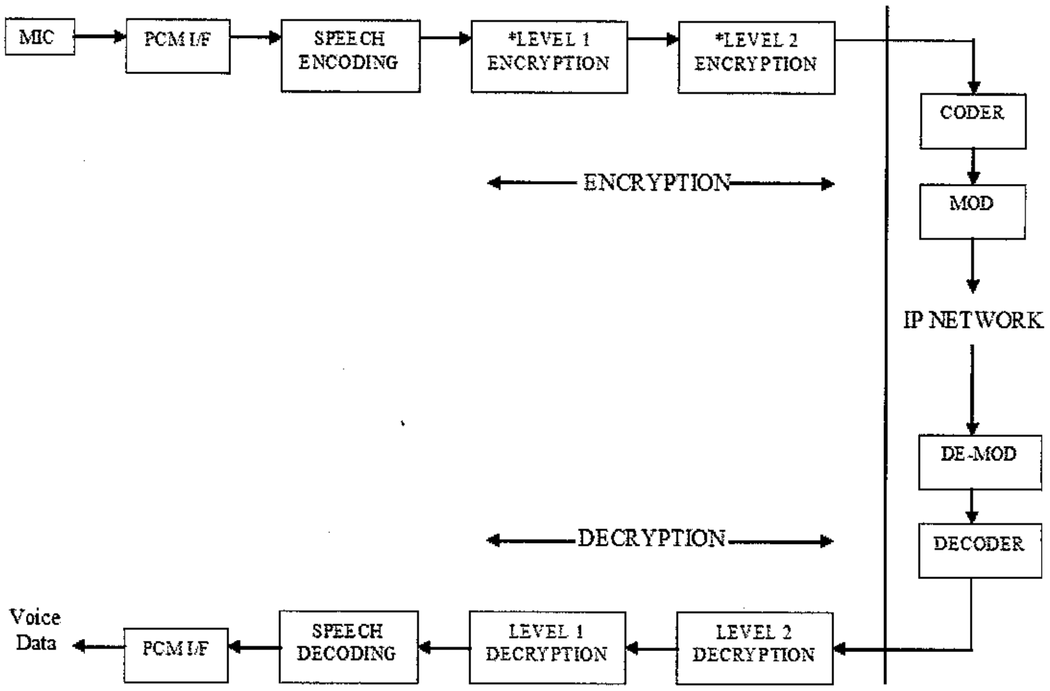


Fig. 4.1 *Block diagram of proposed security scheme for VoIP*

The scheme in the block diagram above illustrates that the proposed level one encryption which is two's compliment overflow nonlinearity encoder-decoder and level two encryption which is the bit swapping are inserted in the traditional VoIP data flow between the speech CODEC and the channel CODEC. These operations are controlled by a chaotic bit string generator based on the chaotic system of a one-dimensional logistic map. Using the chaotic bit string in the level one and two encryption-decryption techniques possesses the features of low computational complexity and high security. In the following the author describes the system formally.

Level one encryption is the two's compliment encoder-decoder and the second level is the bit swapping function which has been highlighted and discussed in Chapter 2 and its algorithm and implementation in Chapter 3.

To summarize the operation of both, the encoder-decoder in Figure 3.4, replaces the data frame interleaving. The bit swapping function cannot be used as a standalone encryption-decryption scheme and is not secure enough. Hence the introduction of the first level of encryption before bit swapping. The diffusion technique is used to achieve a certain level of secrecy or privacy [2]. Diffusion is where the characteristics of the encoded signal are independent of the information signal.

In the encoder the information signal i(t) has be encoded by the operation φ(i, k) using the key signal k(t) where the operation is based on a two's compliment overflow linearity function. The operation is realized by complex scrambling using signal transformations. The mod function is illustrated as

$$\text{mod}(x) = x - 2 * \text{floor}((x+1)/2)$$

which is the two's compliment nonlinearity. To have the identical key signals, the chaotic encryption scheme is realized based on a feed back-feed forward control of the encoder and the decoder by the encoded signal as in Figure 3.4 also known as the self-synchronizing structure which does not need an extra synchronization signal.

The coder states that s (t-j) based on equation (4) and (5) in Chapter 3, is simply time-delayed versions of the encoded signals of s (t). Lets say for a 3rd order encoder-decoder pair, to encode sample 7 which is i (7) where n=3 (order), thus the equation in (1) would be

$$s(7) = \text{mod}(i(t) + (k(1)\, s(6) + k(2)\, s(5) + k(3)\, s(4))$$

The range of s (t) is always limited to [-1 1] due to the modulo-property.

The self-synchronizing scheme leads to a dynamical system with chaotic behavior [2]. The sensitivity of the system to the initial condition and the key value and number of key used, has it made it a suitable candidate to be implemented in this VoIP security system.

Replacing the interleaving block in the HDSP scheme with this encoder-decoder pair has also another notable advantage. The interleaving techniques groups frames of perhaps 10 and swap the frames with one another. This increases processing delay as all 10 frames need to be complete before swapping takes place. The encoder-decoder pair on the other hand, handles samples and receives them in a continues manner. The processing delay is dependant only on the nth order of the system. If it is of order 16, there would be a delay of 16 samples which is within microseconds and not perceivable by the human ear. The speech is also continuously scrambled.

However, the signal s (t), which controls the key generator, is accessible and hence can be used for cryptanalysis. This is one of the main factors of why a second level of encryption, the bit swapping is introduced. Thus as the samples s (t) is sent out of the encoder, it will be further encrypted locally within the sample itself before being sent to the network.

The intra frame encryption, the pixel value transformation which includes two operations of bit swapping and XOR operations based on the chaotic bit string b(0),b(1).....b(n) which transforms the original pixel value to an encrypted one. This occurs locally within the sample. This data encryption approach has the features of low computational complexity. A factor, which affects processing time, is the number of iterations used. Increasing the number iteration by a factor of two will double the required processing time. Hence, instead of iterating the map to the number of samples which is not necessary, the logistic map is iterated every two values. Two values of x (n) and x (n+1) will produce 32-bit chaotic bit string. Each 4 bit will be assigned to the first 8 incoming samples. Once this is completed, the next two values are iterated.

If a scheme involves with pixel permutation, XOR operation will be included to enhance the security. But based on the bit-swapping algorithm in Chapter 3, the XOR operation on each pixel is done outside the permutation looping. In short, after the permutation process is repeated for a said number of iterations, each pixel value is XORed with the generated chaotic bit string.

4.2 IMPLEMENTATION OF PROPOSED SECURITY SYSTEM FOR VOIP IN C

To verify the performance of the two's compliment overflow nonlinearity encoder-decoder and bit swapping done locally within the sample in the proposed scheme, the author performed a software simulation using the Visual C++ on test voices. Figure 4.2 illustrates the simulation environment.

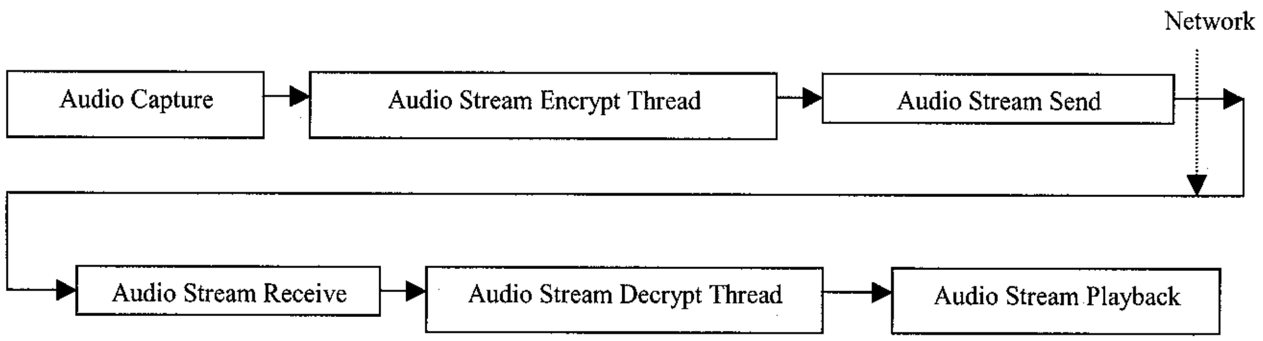


Fig 4.2 *Block diagram of encryption-decryption simulation environment using the proposed two-level encryption schemes*

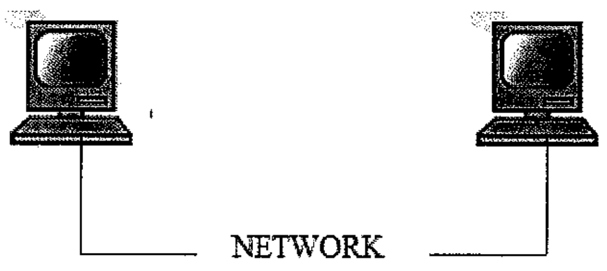


Fig. 4.3 *Block diagram of simulation environment*

The proposed system will be simulated on a setup such as the one in Figure 4.3. Two computers running the standalone application will be connected via the local area network (LAN).

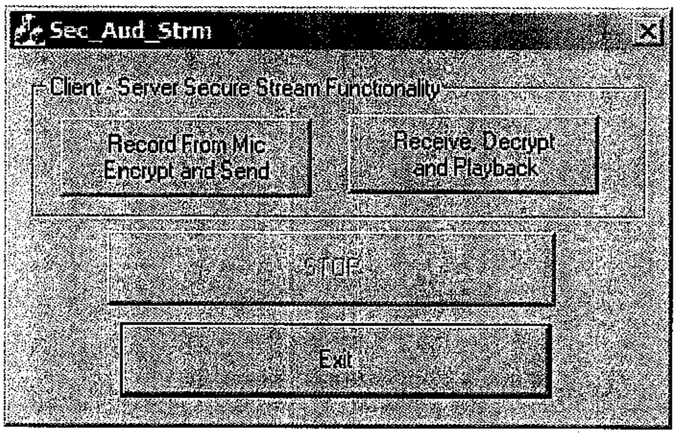

Sec_Aud_Strm
Client - Server Secure Stream Functionality
Record From Mic Encrypt and Send
Receive, Decrypt and Playback
STOP
Exit

Fig. 4.4 *User interface*

Each application will have a dialog box as the user interface as such in Figure 4.3. The receiver will click on the 'Receive, Decrypt and Playback' button and awaits the sender's response. The sender clicks on the 'Record from Mic' button and speaks into the microphone. The voice input will be captured every say for every 1/25th of a second and stored into a global buffer. A sampling rate of 11025 KHz at 8 bits per sample is used which means that 11025 KHz / 25 will give 441 bytes per network packet. These parameters are user-definable in the codes audio_param.h attached in Appendix E and are used for testing purposes.

As said earlier, every 1/25th of a second, it will record from microphone and store the samples byte by byte it into a global buffer. The buffer will consist of 25 packets each containing 441 bytes. Each slot will have a flag. Thus if, slot one has captured all 441 bytes, the flag will be set to 1. The encryption thread will check the flag status. If 1, sample by sample (8 bits per sample) will be sent to the first level of encryption which is the two's compliment overflow nonlinearity encoder. Once encrypted, it will store the encrypted output in a local buffer. Then, byte by byte will be encrypted using the bit swapping function. Once encrypted, the samples will be packetised in TCP packets to the receiver side. At the receiver end, the decryption thread will receive the packet and decrypt in reverse order, second level decryption followed by first level. Once decrypted, it will store samples into the global buffer for playback.

Only one thread is used on both sides of encryption and decryption. To enhance the efficiency of the system and decrease processing delay, there can be two encryption threads for both level of encryptions so that the encryption of both levels can be done simultaneously and no need for intermediate buffers. This is to be applied to the decryption side as well.

The size of frame/packet should be carefully selected so that it has lower latency and overhead processing. The smaller the packet size, the better error recovery and will reduce the end-to-end delay in a VoIP system.

The TCP transport protocol is not suitable for the VoIP applications because of its complex retransmission mechanism. For this reason, all the VoIP applications at present transmit via the UDP transport protocol. However, assuming that the delay is minimal and less than 240ms, TCP is used in this transmission because the proposed system does not offer a framework for packet loss. Hence, using UDP may cause the packet loss to be critical resulting in the recovered signal or voice on the receiver end to seem as if it was not decrypted completely. Hence for simulation purposes and to demonstrate the workability of the two-level encryption scheme by assuming minimal delay on the LAN, TCP was adapted into the system.

4.3 SIMULATION AND ANALYSIS RESULTS

The AES encryption scheme is used to compare against the symmetric chaotic encryption scheme and will be discussed in the following sections.

The MATLAB implementation of the AES encryption was obtained from http://buchhloz.hs-bremen.de/aes/aes.htm. This encryption scheme is used to compare against the symmetric chaotic encryption scheme and will be discussed in the following sections. These m-files were compared to the AES standards to ensure its validity. This implementation of AES is fully operational but only does not optimize speed. For each of the first Nr-1 iterations or rounds, it performs a substitution operation called Sub Bytes on State using an S-Box; perform a permutation Shift Rows on State; perform an operation Mix Columns on State; and perform AddRoundKey.

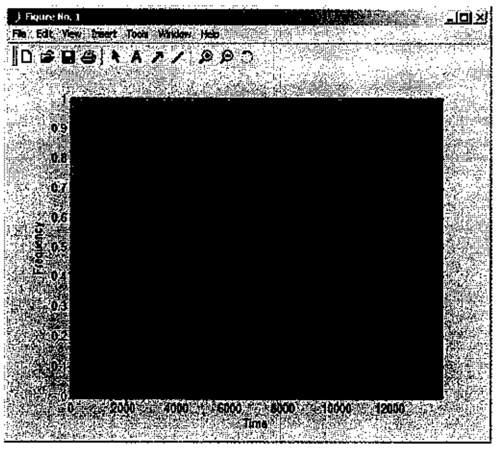
The key for the chaotic encoder-decoder had been synchronized and obtained by iterating the logistic map for n times where n is equivalent to the number of keys used. For comparison with AES, the keys for the chaotic encryption have been set to 128 bits as equivalent to the AES key length.

4.3.1 Spectrogram Analysis

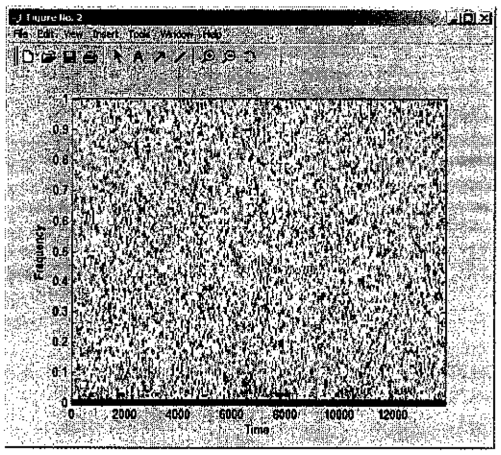
The spectrogram function in MATLAB divides a long signal into windows and performs a Fourier Transform on each window. It basically shows the reader what is the frequency content of the signal at any point of time. It produces a pseudo color display of spectral energy with frequency on the vertical axis and time on the horizontal axis. The colors represent the most important acoustic peaks for a given time frame, with red representing the highest energies, then in decreasing order of importance, orange, yellow, green, cyan, blue, and magenta, with gray areas having even less energy and white areas below a threshold decibel level. For a given spectrogram, *S* the strength of a frequency component *f* at a given time of *t* is represented by the darkness or the color of the corresponding point *S (f, t)*.

The spectrogram in Figure 4.5 in the following page illustrates the voice signal "*Hello. Testing 1, 2, 3.*" An experienced spectrogram reader would be able to identify the words from the patterns in the image above. The vertical line patterns in red and blue could depict the sounds of the certain alphabet at the end of a word or syllable. The other speech sounds or *phonemes* are equally distinctive in their shapes. It is not possible to obtain or read the phonemes from a normal speech waveform. However, when analyzing the frequency content of the waveform, a spectrogram is produced as above which can be deciphered. Nevertheless, the distinctive shapes, patterns and colors need to be present in order to be able to identify the words.

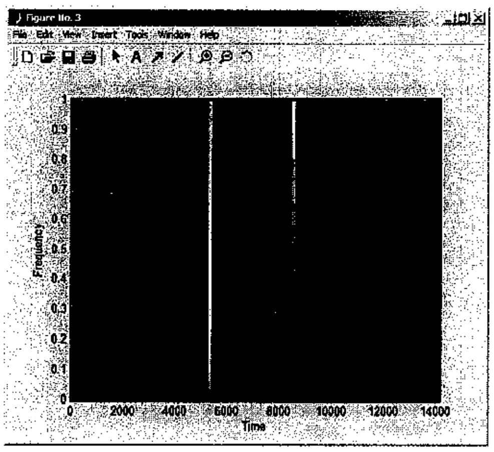
The spectrogram in Figure 4.5(b) shows that the signal has been distorted and also displays only no significant frequency or energy content. Hereby, no distinctive shapes or patterns are seen and the rows of frequency content are not at all visible for a code cracker to make out the speech that is being said. It is impossible to distinguish or deduce the original signal from it since there are no distinctive significance between the original signal and the encrypted one.



(a)



(b)



(c)

Figure 4.5 *Spectrogram of (a) original voice signal, (b) encrypted signal and (c) decrypted signal using chaotic encryption scheme*

The decrypted signal as in Figure 4.5(c) is not 100% similar to the original signal though theoretically it should be the same. The author suggests that the blue line in the original signal and the yellow line in the decrypted occurs at the same instance for any recording, and may be caused by internal hardware. However, when the decrypted signal is played back, it sounds exactly as the original signal with no distortion or noise corrupting it.

4.3.2 Comparison between Advanced Encryption Standard (AES) and Symmetric Chaotic Encryption

To compare AES and chaos, a speech signal *"Hello.wav"* was used and the results are shown in the figures below and both encryption schemes using 128-bit key.

Figure 4.6 illustrates the speech signal that was fed into both the AES and chaos encryption scheme. The signal that was sent it is identical and thus only the method of encryption will hence produce different results in the encrypted as illustrated in Figure 4.7 (a) and (b).

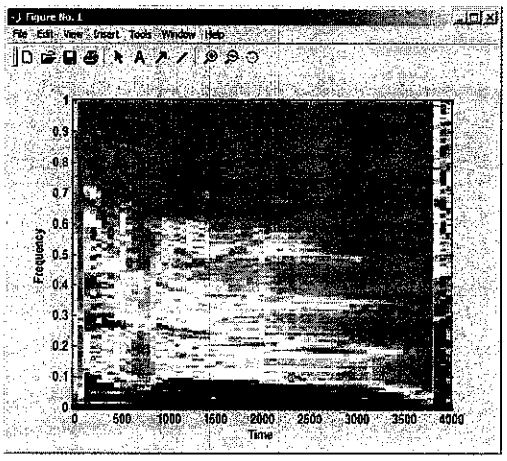
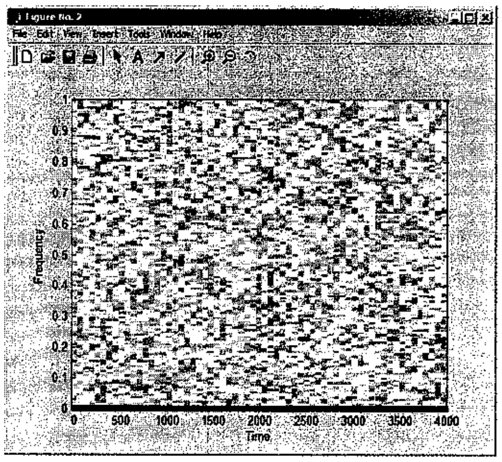
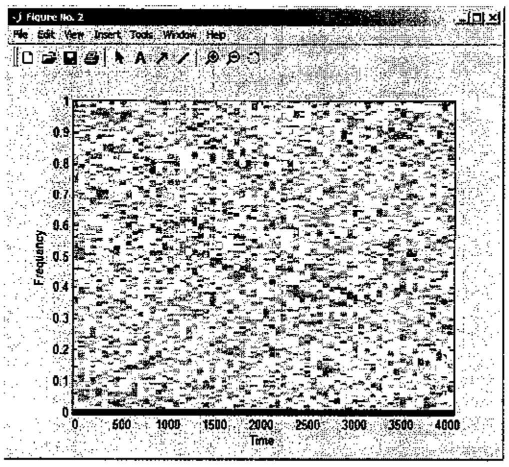


Figure 4.6 *Spectrogram of original voice signal*

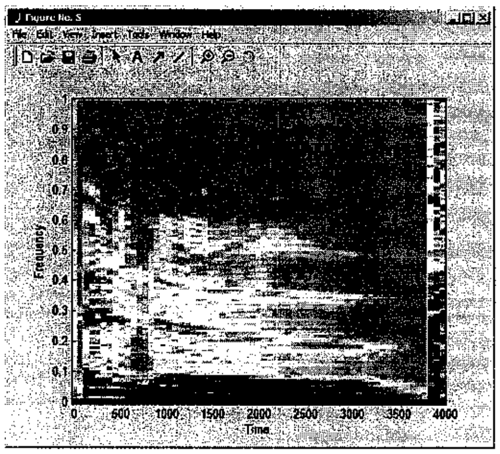


(a)

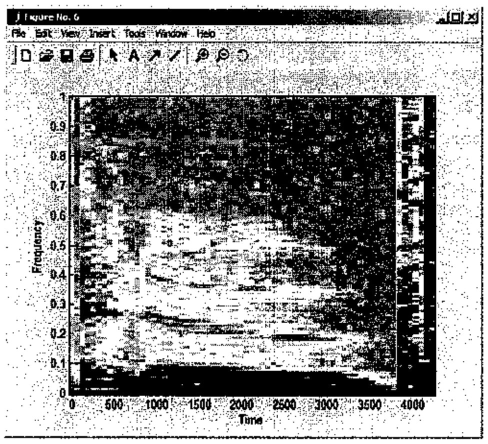


(b)

Figure 4.7 *The encrypted signal displayed using a spectrogram for (a) 128-bit AES and (b) Chaos*



(a)



(b)

Figure 4.8 *Spectrogram of decrypted voice signal using (a) AES and (b) Chaos*

The encrypted signal for AES in Figure 4.7 (a) shows all kinds of frequency content and is clearly distorted if compared to the original one. As mentioned earlier, for any speech signal to be read from the spectrogram, it has to have distinctive shapes and patterns, which is clearly not the case for this image of the spectrogram. Even Chaos in Figure 4.7(b) displays the same kind of characteristics; randomly distorted with no distinctive patterns and the rows of frequencies is not visible as the original image. Only the energy content in both this these spectrograms differ. If the key values were changed, the cipher text obtained was totally different from one another though resulting in the same decrypted spectrogram. Since this is the portion to be viewed by the code cracker, it would be hard to pick out any kind of frequency variation and read out the *phonemes,* as there is no significant resemblance to the original spectrogram.

For AES, since the decryption process is just an inverse of encryption, thus the original signal is hence recovered with not the slightest changes to it. For chaos however, there is a slight shift to the right in the decrypted signal. Based on the Figure 3.4, for any given signal encrypted by chaos, the decrypted portion of the signal experiences a slight shift to the right or in other words a delay due to the delay system used in the encryption-decryption block. But the delay is not obvious nor is it perceivable to human ears.

Using another wav file produced the results as illustrated in Figure 4.9 and Figure 4.10(a) and (b) and Figure 4.11(a) and (b).

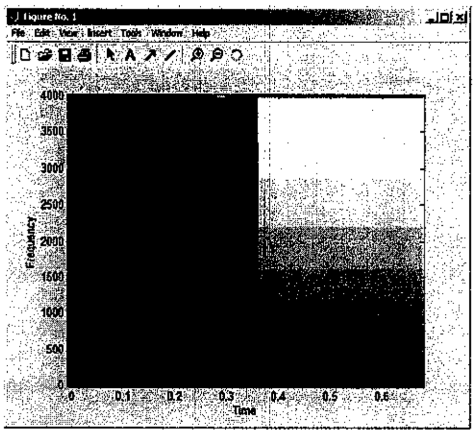
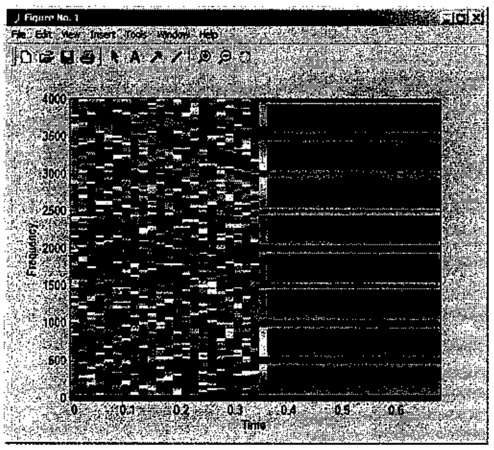
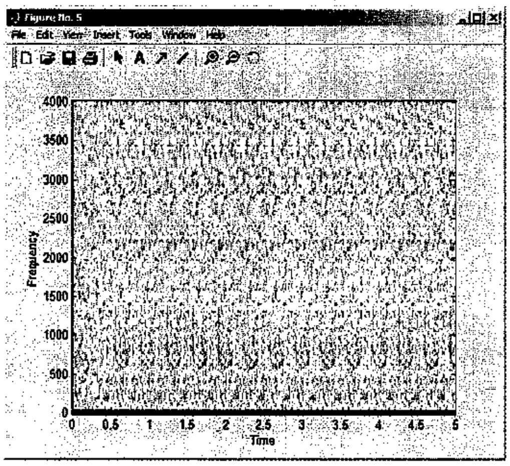


Figure 4.9 *The original file being fed to the AES and Chaos*

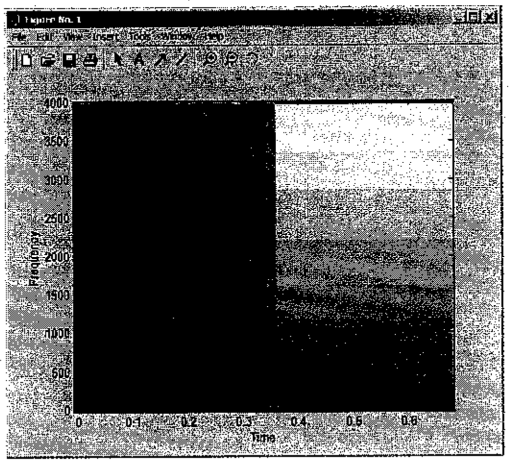


(a)

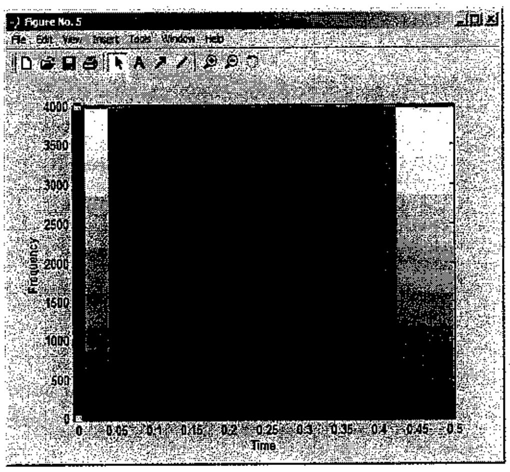


(b)

Figure 4.10 *The encrypted signal displayed using a spectrogram for 128-bit (a) AES and (b) Chaos*



(a)

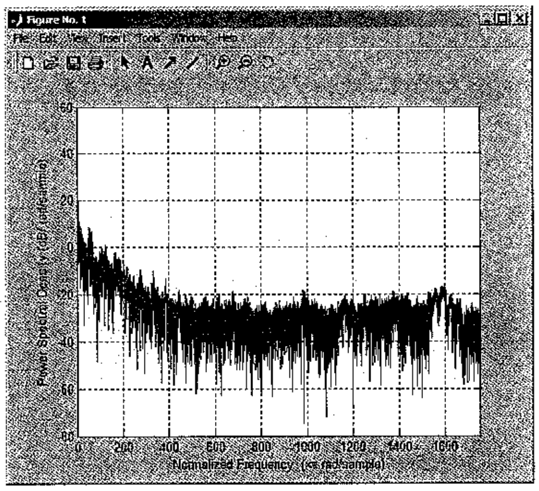


(b)

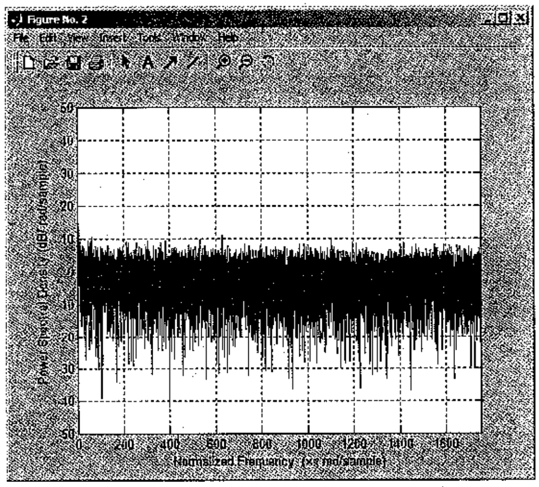
Figure 4.11 *The decrypted signal for (a) AES and (b) Chaos*

4.3.3 Power Spectral Density and Histogram Analysis

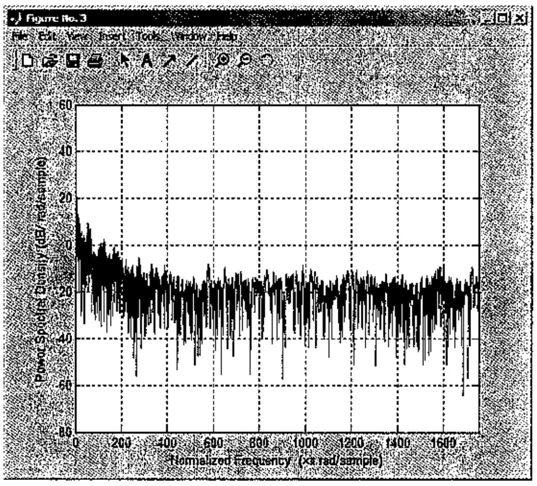
The power spectral density is to describe the distribution of the power contained in the signal over frequency. According [32], the estimation of this power spectral density is useful in detecting signals buried in wide-band noise.



(a)



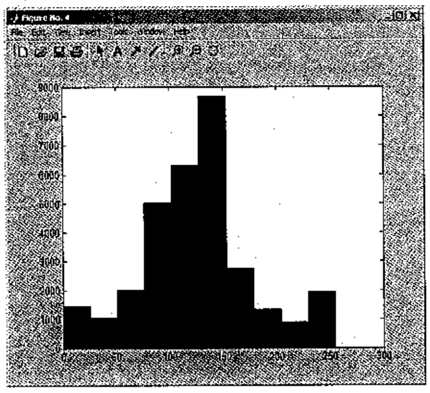
(b)



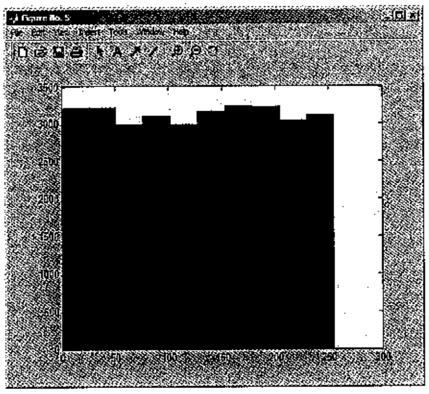
(c)

Figure 4.12 *Power spectral density plot of (a) original voice signal, (b) encrypted signal and (c) decrypted signal using chaotic encryption scheme*

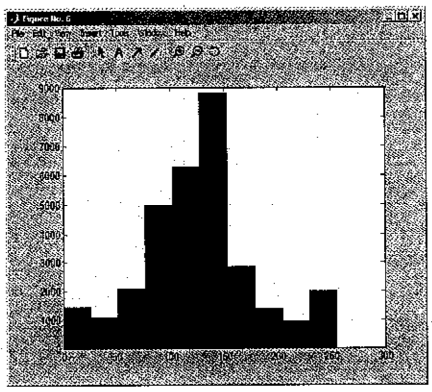
The noise like signal in Figure 4.6(b) is close to that of white noise. This is because, it portrays the characteristics of white noise in terms of being uncorrelated, completed random and has a relatively flat spectrum. The closer it is to white noise, the better. This is because white noise is uncorrelated meaning there is no implication of the relationship between the original signal and the noise produced. Also, detecting the significant frequencies from a relatively uniformed signal as in Figure 4.6(b) is a highly difficult task. Hence recovering the original signal would take more time and effort for the code cracker. The case of uniformity can be further illustrated with a histogram analysis. The histogram block computes the frequency distribution of the elements in each column of the input, or tracks the frequency distribution in a sequence of inputs over a period of time. The histogram values represent the frequency of occurrence of the input values



(a)



(b)



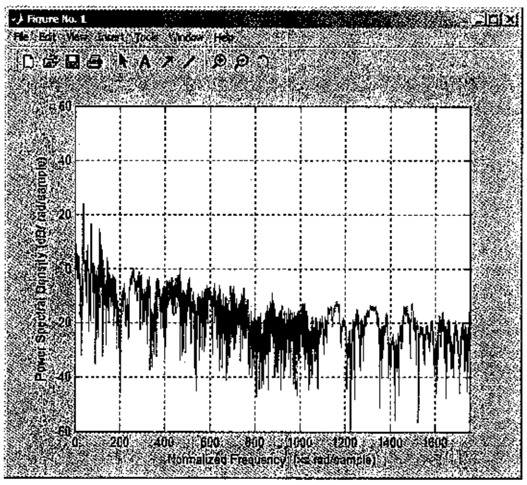
(c)

Figure 4.13 *Histogram plots of (a) original (b) encrypted and (c) decrypted voice signal using chaotic encryption scheme.*

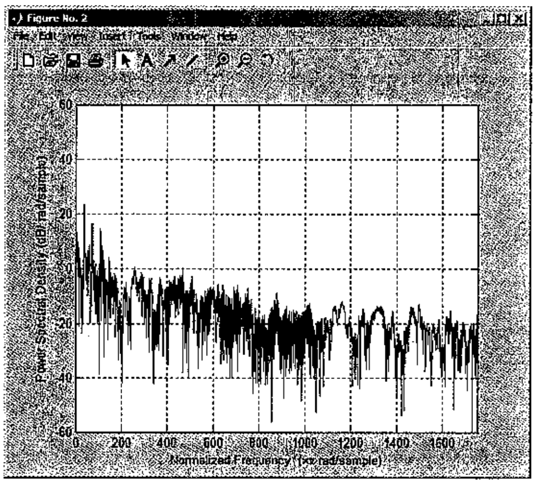
The original signal in Figure 4.13(a) is seen to have significantly high and low frequencies unlike the encrypted one in (b) for which the frequencies occur more closely to one another and there are no significantly high and low levels of frequencies.

From the figure in 4.13(b), the histogram plot is seen to be relatively uniform. It is harder to detect the original frequencies from uniformity as depicted in (b) and less information is perceived from a poor intensity distribution.

4.3.4 Comparison between Advanced Encryption Standard (AES) and Symmetric Chaotic Encryption



(a)



(b)

Figure 4.14 *Power spectral density plot of voice signal using (a) AES and (b) Chaos*

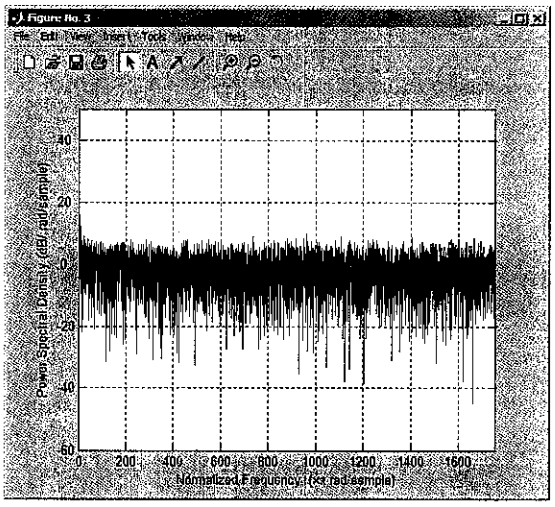


Figure No. 3
File Edit View Insert Tools Window Help
Power Spectral Density (dB/rad/sample)
40
20
0
-20
-40
-60
0 200 400 600 800 1000 1200 1400 1600
Normalized Frequency (×π rad/sample)

(a)

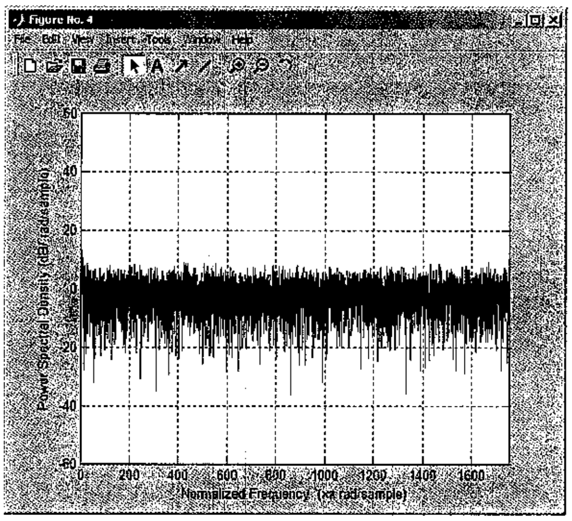
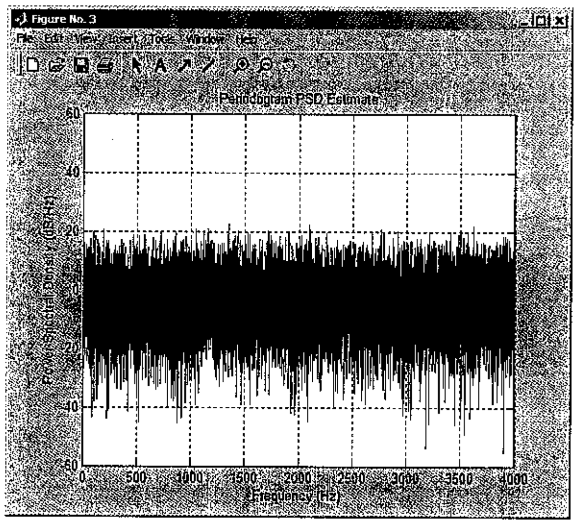


Figure No. 4
File Edit View Insert Tools Window Help
Power Spectral Density (dB/rad/sample)
60
40
20
0
-20
-40
-60
0 200 400 600 800 1000 1200 1400 1600
Normalized Frequency (×π rad/sample)

(b)

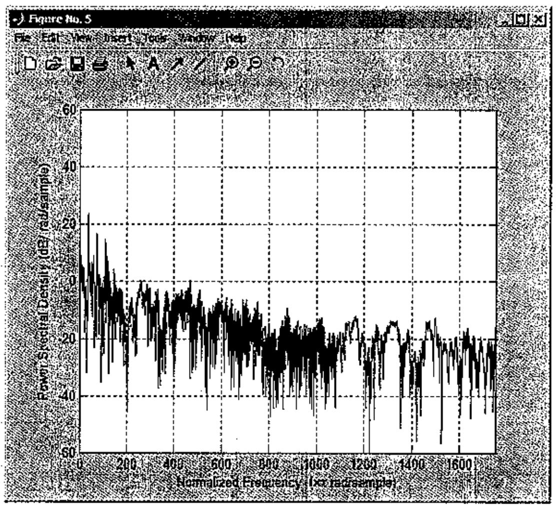


(c)

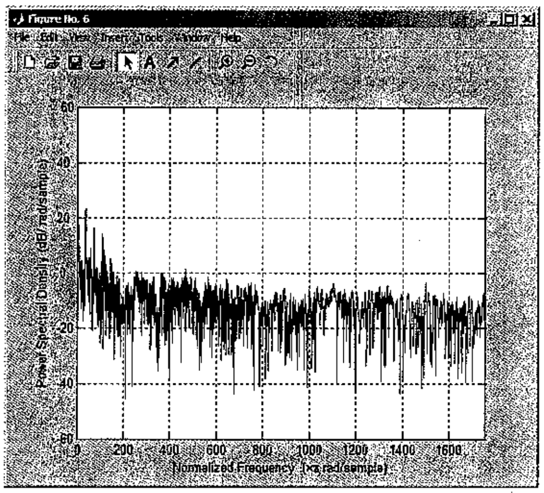
Figure 4.15 *Power spectral density plot of encrypted voice signal using (a) AES, (b) and (c) Chaos using different key values*

Based on the power spectral density plots gathered on both the encrypted signal and original signal using chaotic encryption, the noise signal or encrypted one, there is no significant similarity in pattern between the original signal and the encrypted one. The original signal depicts a typical voice signal, which decays at the end as the voice gets cut off.

Both encrypted signals using AES and Chaos are randomly distorted using different methods, producing almost similar power spectral density plots. However, the encrypted data has been spread quite uniformly over the frequency range for AES using 128 bits and for Chaos in Figure 4.14(b) and (c) which uses different key values. The noise like signal is close to that of white noise. This is because, it is uncorrelated, completed random and has a relatively flat spectrum. Spreading the data uniformly is a good thing as it makes it harder for code cracker. He or she has to have a device to listen to all frequency range. Listening to all freq range is also difficult as there will be other frequency signal like TV, Radio, and mobile phone. Having said that, comparing AES in (a) and Chaotic in (c), the magnitude of Chaotic (10^-2 to 10^2) is higher than the magnitude of AES (10^-1 to 10^1). This means the chaotic signal carried more power that represents its content over the same band of frequency.



(a)



(b)

Figure 4.16 *Power spectral density plot of decrypted voice signal using (a) AES and (b) Chaos*

Once again the case of uniformity can be further explained using the histogram plots.

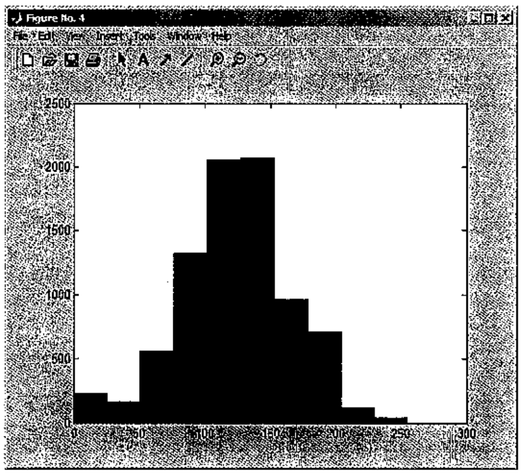
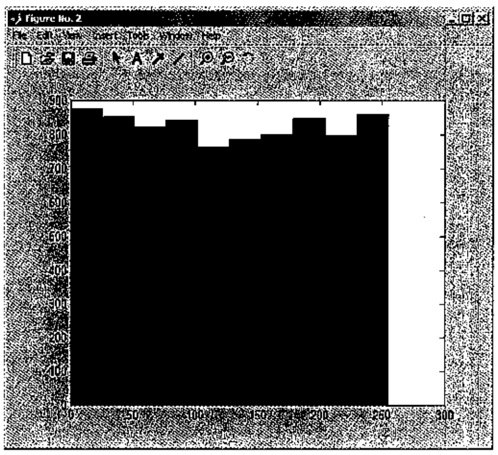
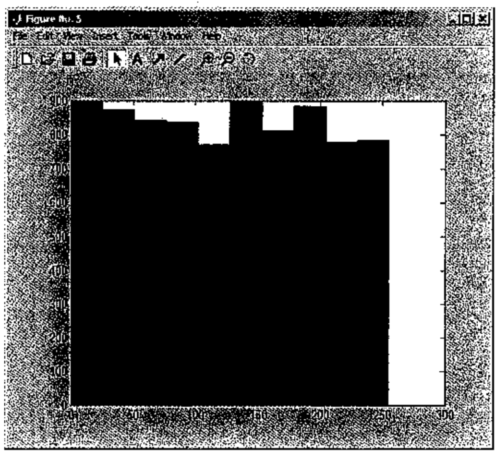


Figure 4.17 *Histogram plot of original voice signal*

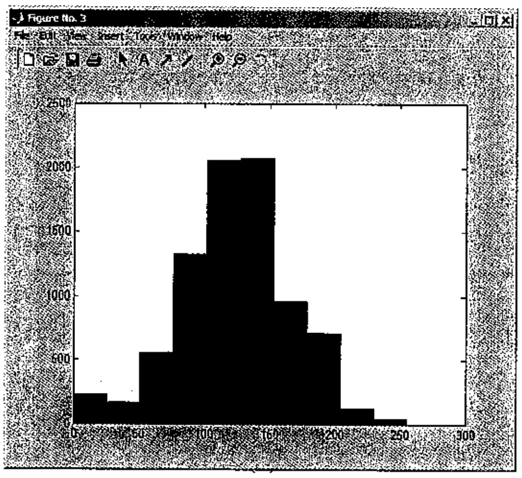




(a) (b)

Figure 4.18 *Histogram plot of encrypted signal using (a) AES and (b) Chaos*

As observed, the histogram plots from the output of encryption in Figure 4.17 are almost similar and uniformly distributed for both AES and Chaos encryption scheme. Thus this is also indicates a certain potential possessed by a chaos system to produce encryption results as similar to that of the existing standards such as AES.



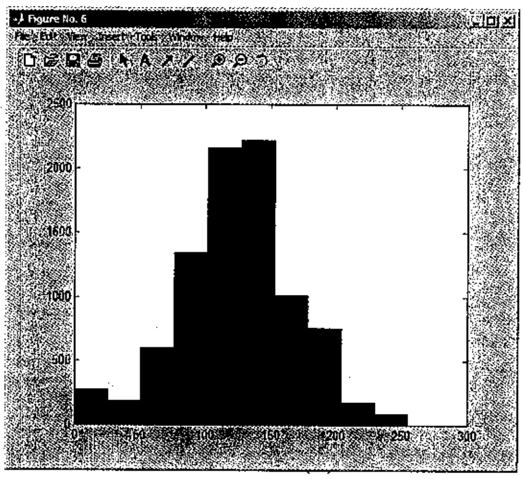
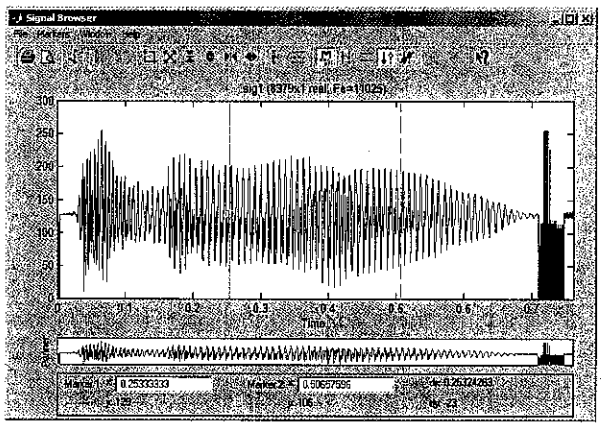


Figure 4.19 *Histogram plots of decrypted signal using (a) AES and (b) Chaos*

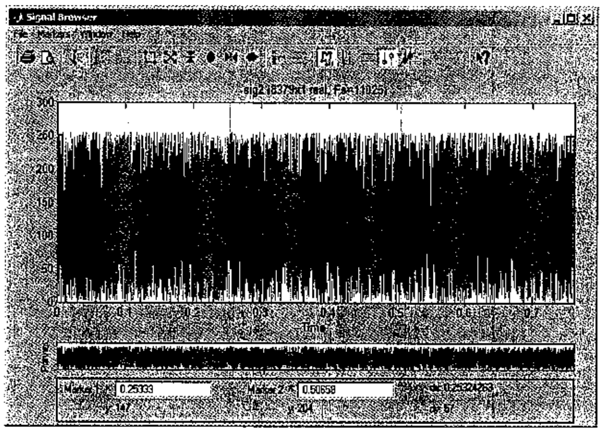
4.3.5 Time-Waveform Analysis

Using a wave file "*Hello.wav*", the illustrations in Figure 4.20 and 4.21 shows that the time waveforms of the encrypted signal from both the AES and chaotic encryption scheme were noise like broadband power spectrum. The output was completely erratic and when played back, was completely unintelligible.

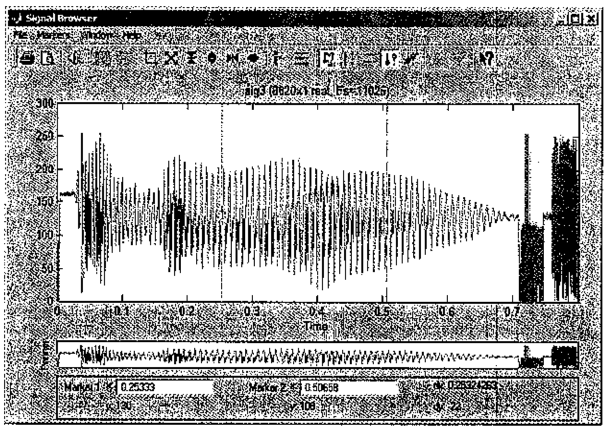
The author observed a DC shift in the decrypted signal obtained from the chaotic encryption when compared to the original signal. Since this a trivial issue and was identified only later in the testing stage, the author is currently debugging the codes to rectify the error.



(a)

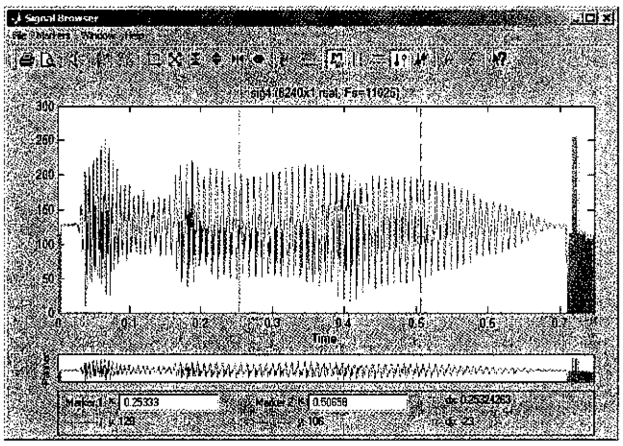


(b)

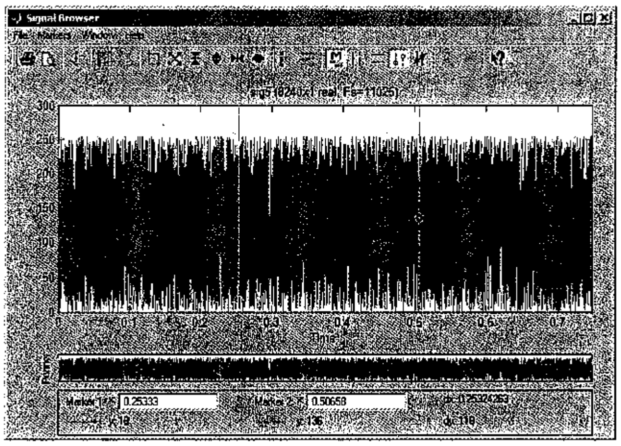


(c)

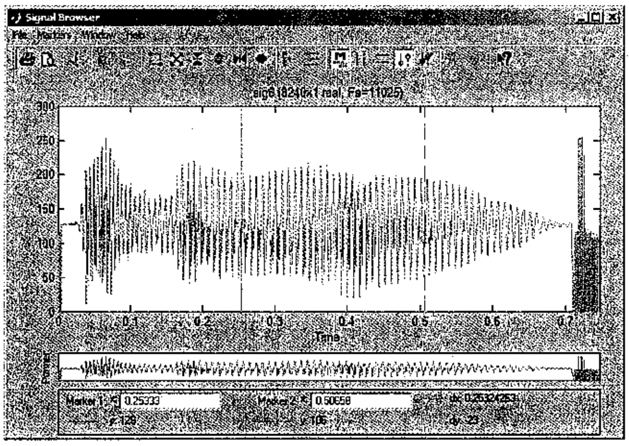
Figure 4.20 *Plot of (a) Original signal, (b) encrypted and(c) decrypted signal using AES encryption scheme*



(a)



(b)



(c)

Figure 4.21 *Plot of (a) Original signal, (b) encrypted and(c) decrypted signal using Chaos encryption scheme*

4.4 CONCLUSION OF ANALYSIS

From the observation and results, it can be deduced that chaotic encryption schemes have a good future and have produced attractive results, which can be further researched and enhanced.

Though both the AES and Chaos based encryption schemes deploy random characteristics to encrypt the incoming data be it voice, text or video, but the outcome of the encryption still differs due to the operation or method used to encrypt the data.

From the author's understanding, on "what makes a good encryption algorithm", though not complete, is outlined as below:

- Iterations required per sample
- Speed of encryption and decryptions
- Error propagation (when you iterate something, i.e. current output depends on the previous output etc, they tend to produce error propagation)

Thus, these are the criteria to look into when comparing two encryption schemes.

Encryption strength would relate to the time it would take to decrypt not having the right key. In this case, the algorithm is unknown and attempts are being made to detect an encrypted signal disguised as noise. Having detected the presence of a signal, an attempt to decrypt it will depend on whether the code cracker has access to the encryption algorithm and has to figure out the key, or on whether the encryption method has to be worked out as well. Looking at AES and Chaos based encryption and the corresponding results, the noise signal obtained is completely random, and thus with a power density spectral which is relatively uniformly distributed, it does not indicate any similarity in pattern corresponding to original signal which could provide any possibility to decrypt it.

Be it for audio, text, video or image, based on [1] and [10], there are no known cryptanalysis attacks to have worked on AES and due to its operations especially the key expansion operation which fulfills the design criteria to reduce the symmetric characteristics of the encryption scheme and round transformations; it is highly resistive against attacks especially as the number of rounds (iterations) increases.

Chaos on the other hand, possessing characteristics of randomness, stochastic and mixed-ergodic, enhances the security of the encryption process by the means of a chaotic dynamical map. With reference to the results obtained, it can be said that the chaotic encryption scheme is quite attractive to be used as replacement for currently existing standards. The cipher text is seen to be a very erratic plot compared to the plaintext. It was observed that for audio encryption, the system shows great sensitivity to initial conditions, changes in number of keys and changes in key values. Using a single channel voice data as input, the cipher text obtained was a noise-like sound.

It was observed that, with the key value zeroes; the encryption did not have any effect on the signal. But when the key values were greater than zero, the cipher text was successfully obtained. This has to be looked into, as sometimes, the key values are picked at random and may give an all zero value. To avoid this, the logistic map is used to obtain chaotic values between 0 and 1.

Based on the author in [1], the fractal dimensions of the original and encrypted voice should range from 1.7 to 1.9. Since the maximal fractal dimension for a one-dimensional curve is 2.0000, all the encryption results is said to be in a state of chaos.

In [4], it is mentioned that statistical analysis needs to be conducted on the publicly available encrypted signal at the encoder-decoder. The principle behind this is to create statistical measure based on the encoded signal and key parameters and then try to estimate the keys. This is much similar to the exhaustive scheme used in cryptanalysis although the former is supported by statistical data. Although it is easy to break an encryption using this type of analysis, it was pointed out by the author that choosing a larger key set from a larger set would increase the time taken to estimate the keys. The table below illustrates the number of days taken to estimate the key with an accuracy of 0.1 in coefficients.

Table 4.1. Time taken to estimate key based on order deployed in encoder-decoder structure in Figure 3.4.

Order (n)	Time
2	25 s
3	8.3 min
4	4.2 hr
5	3.5 days

The issue of security for this scheme still needs to be resolved since this implementation has not dealt with the level of secrecy of the encryption scheme.

The secrecy level of this chaotic encryption though not greater should be equally as good as the existing standards such as the Advanced Encryption Standard (AES). Though, the level as of encryption for chaos is greater with increasing number of keys, however, a better solution is to have a smaller number of keys but able to produce the same level of security and secrecy. The AES encryption provides this ability being able to only use one key to encrypt and decrypt but with the choice of 128, 192 and 256 bits. For this analysis, only the 128 bit key was used. The encoder-decoder had to have an order of 8 delays to produce a set of 128-bit key. With the bit swapping algorithm aiding the encoder-decoder, the encrypted signal produced by chaos is almost similar to the one produced by the AES encryption scheme as observed in the illustrations above.

Pertaining to the issue on the security of the symmetric key, an alternative would be to integrate the chaos encryption with Public-Key Infrastructure based encryption such as the RSA.

To enhance the security of symmetric chaotic encryption, the integration between a symmetric encryption and PKI based encryption also using chaos would prove to be a work around to reduce the probability of the key being known to a code cracker. The symmetric key or keys used to encrypt the plaintext is encrypted using the public key before sending it over the transmission line to the receiver. Only the receiver is aware of his or her private key and hence will be able to decrypt the symmetric key, obtain it and use it to decrypt the cipher text. In this case, even the receiver need not know of the symmetric key to be used before hand.

If this concept is successfully applied to chaotic encryption, hence the security will be further heightened.

CHAPTER 5

CONCLUSION and RECOMMENDATION

Cryptography provides a solution to the problem of information security and privacy. Encryption is the correct method to implement confidentiality for Internet traffic. VoIP technology is progressing admirably, but certain drawbacks have been indicated such as packet loss, propagation delay, jitter, unreliable IP networks, and vulnerability to attacks by Internet hackers. Thus, it is essential to perform data encryption on voice transmitted over the Internet while preserving the quality of the voice from packet loss.

Chaos, a relatively new science is open for exploitation in broad ranging fields. Chaos has attracted much attention in the field of cryptography for private and secure communications due to its deterministic nature and its sensitivity to initial values. Such properties mean that chaos has certain potential in creating a new way of securing information to be transmitted or stored. It is a simple method of encryption and can be achieved by iteration with a high level of security.

This project allowed the author study, develop and implement a chaotic encryption scheme. The project proposes a chaotic encryption scheme based on a chaotic system which will preserve data security for real-time voice data encryption. This project work is based on the scheme in [1] and proposes a two-level encryption decryption scheme based on the two's compliment overflow nonlinearity encoder-decoder and bit swapping. This scheme will apply to a VoIP network.

With regards to the objectives outlined in Chapter 1, which is to implement a two-level encryption-decryption scheme for voice data, this project is considered successful. The author has successfully drawn up a new scheme and tested it across the local area network. The results and observations has been laid and discussed in Chapter 4 as well as compared to the AES encryption scheme. From observations, the encrypted signal of the chaotic encryption scheme is almost similar to that of the AES encryption.

Due to time constraint however, the author, had not dealt with the analysis of the security level of the scheme. However since the scheme was based on the [1] and [2], this issue has been solved by referring to arguments presented in the two, which has also been discussed in Chapter 4. However, it is to be noted that these arguments cannot be one hundred percent be used to validate the algorithms strength and level of security. It is hence recommended that future work will deal with this issue and proper cryptanalysis measures be taken to test and validate the algorithms implemented. To warrant an algorithm as secure, it has to be proven to resist attacks. Mathematical functions such autocorrelation that for a highly random signal produces a delta function and zero elsewhere and fractal dimensions calculations as in [1] should also be included.

Based on the discussion in Chapter 4, the codes provided in the appendices can be further optimized by increasing the number of threads to encrypt and decrypt as to decrease the processing delay. Also, the packetization using TCP should be replaced by UDP before implementing in a WAN or MAN based environment. However, the scheme should be enhanced and a framework for packet loss must be considered.

Since the codes have been written as a stand-alone C program, it can be further used for FPGA implementation and for the use of smart card based solution. From the author's own understanding and reading, chaotic encryption can be realized to be a powerful tool in cryptography if exploited and researched further. This chaotic encryption scheme can be further analyzed, fully realized and implemented on hardware or other areas, which can be explored further.

In short, the author has managed to implement a two-level encryption-decryption scheme for voice data. However, the strength of this scheme needs to be further analyzed, tested and validated.

REFERENCES

1. J.I. Guo, J.C. Yen and H.F Pai. "New voice over Internet Protocol technique with hierarchical data security protection," *IEEE Proc. Visual Image Processing, vol.149, No.4, August, 2002.*
2. Goltz, M. Kelber, K Schwarz, W. "Discrete Time Chaotic Encryption Systems," *IEEE Trans. Circuit Sys. I, Fundamental Theory and Applications, vol.44, No 10, October. 1997.*
3. Dachselt, F. Kelber, K Schwarz, W. "Discrete Time Chaotic Encryption Systems-Part III: Cryptographical Analysis," *IEEE Trans. Circuit Sys. I, Fundamental Theory and Applications, vol.45, No 9, September.1998.*
4. F.Beritelli, E. Di Cola, L.Fortuna and F.Italia., "Multilayer Chaotic Encryption for Secure Communications in Packet Switching Networks," *IEEE.2000*
5. Princy Mehta and Sanjay Udani. *Voice over Internet Protocol: Sounding good on the Internet.*
6. Bur, Goode. "Voice over Internet Protocol," *IEEE, vol.90, No. 9, September. 2002.*
7. Hun-Chen Chen, Jui-Cheng Yen, Jiun-In Guo. "A new hierarchical chaotic Image encryption and its VLSI realization", *IEEE Proc. Vis. Image Signal Process, 147,(2), pp 167-175, 2000.*
8. WU, C.W, and Rulkov, N.F, "Studying Chaos via 1-D maps-a tutorial," *IEEE Trans. Circuit Sys. I, Fundamental Theory and Applications, 40, (10), pp 707-72, 2000.*
9. Mieczyslaw, Jessa, "Data Encryption Algorithms using One-Dimensional Chaotic Map," *ISCAS 2000. IEEE International Symposium on Circuits and Systems, pp 1711-714, May, 2000.*
10. Goce Jamoski and Ljupco Kocarev, "Chaos and Cryptography: Block encryption ciphers based on chaos maps," *IEEE Trans. Circuit Sys. I, Fundamental Theory and Applications, vol. 48, No.2, February. 2001*
11. Ljupco Kocarev, "Chaos based cryptography: A brief overview," *IEEE. 2001*

12. Christopher P. Silva and Albert M.Young. 2000. *Introduction to chaos based communications and signal processing*

13. Gonzalez Alvarez and Shujun Li. 2003. *Cryptographic requirements for chaotic secure communications*

14. He Kangwei and Tan Chaur Lih, *Chaos and cryptography: Applications and Analysis*

15. Terry Rowlands and David Rowlands, *A more resilient approach to Chaotic Encryption.*

16. Ramiro Pablo Costa, *Numerical Investigation of the Logistic Map*

17. Kennedy, M.P., Rovatti, R., Setti, G. 2000. *Chaotic Electronics in Telecommunications*, CRC Press LLC.

18. Brian W.Kernighan and Dennis M.Ritchie, *The C Programming Language*, Prentice Hall

19. J. Cordova Zecena, *Chaotic Dynamical Systems and Their Applications*

20. Nick Whitehead, Michael Overton, Zach Labry, Franklin Hamilton, Brian Leising, *Encrypting Chaos: Fractal Encryption*

21. Douglas R. Stinson 2000, *CRYPTOGRAPHY Theory and Practice*, Chapman & Hall/CRC.

22. Joan Daeman and Vincent Rijmen 2002, *The Design of Rijndael: AES- The Advanced Encryption Standard*, Springer.

23. Elizabeth Oswald, Joan Daeman, and Vincent Rijmen 2002, *AES- The State of the Art Rijndael's Security*

24. Baptista, 1998 , *Chaotic Encryption Techniques*

25. J.Cordova Zecena, University of Arkansas, *Chaotic Dynamical Systems and Their Applications*

26. Irma B.Fernandez, Wunnava V. Subbarao, "Encryption based Security for ISDN Communication: Technique and Applications," *IEEE. pp70-72. 1994.*

27. Michael Welschenbach ,*Cryptography in C and C++*, Apress^{TM}

28. Mike Mcgrath, *C Programming*, Computer Step.

29. http://www.altavista.com/archive/cryptography/chaos_encryption.txt

30. http://www.mathworks.com

31. http://www.mathworks.com/access/helpdesk/toolbox/signal/spectra5.shtml

32. http://www.mathworks.com/access/helpdesk/toolbox/signal/psd.shtml

33. http://www. chipcenter.com/dsp/DSP000531F1.html

34. http://www.nwfusion.com/columnists/2003/1110taylor/voipstillamajorissue.html

APPENDICES

APPENDIX A

Suggested Milestone for the First Semester of the Final Year Design Project

No.	Detail/ Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Selection of Project Topic	▨													
	-Propose Topic: Implementation of a symmetric chaotic encryption														
2	Preliminary Research/Design Work		▨	▨											
	-Introduction														
	-Objective														
	-List of references/literature														
	-Project planning														
3	Submission of Preliminary Report (Proposal)				●										
4	Project Work				▨	▨	▨	▨							
	-Literature Research on chaos and AES														
5	Submission of Progress Report								●						
6	Project work continue								▨	▨	▨	▨			
	-MATLAB Implementation of AES - Analysis and comparison with Chaos														
7	Submission of Interim Draft												●		
8	Oral Presentation													●	
9	Submission of Interim Report														●

● Suggested milestone

▨ Process

APPENDIX B

Suggested Milestone for the Second Semester of the Final Year Design Project

No.	Detail/ Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Project Work Continue		▨	▨											
	-Research on new encryption schemes for VoIP based on a chaos system -Streaming in C and MATLAB -Study HDSP scheme														
2	Submission of Progress Report 1			●											
3	Project Work Continue				▨	▨	▨	▨							
	- Implementation of the proposed two-level encryption scheme in C														
4	Submission of Progress Report 2								●						
5	Project work continue								▨	▨	▨	▨			
	-Implementation of proposed scheme in C -Unit testing and integration testing Analysis of results in MATLAB -Comparison with AES														
6	Submission of Dissertation Final Draft												●		
7	Oral Presentation													●	
8	Submission of Project Dissertation														●

●	Suggested milestone
▨	Process

APPENDIX C

Table 1 A short catalog of chaotic discrete maps.

Map	Definition	Chaotic Regime
Logistic	$x(k+1) = f_{\log}(x(k);A,B) = B(A^2 - x(k)^2) - A$	$x(k) \in [-A,A]$ $3/2 < AB < 2$
Quadratic	$x(k+1) = f_{\mathrm{sqr}}(x(k);A,B) = B - Ax(k)^2$	$x(k) \in [-2/A.2/A]$ $3/4 < AB < 2$
Exponent	$x(k+1) = f_{\exp}(x(k);A,B) = x(k)\exp(B(A - x(k)))$	$x(k) \in [0, \exp(AB-1)/B]$ $AB > 2$
Sine Circle	$\theta(k+1) = f_{\mathrm{cir}}(\theta(k);A,B) = \theta(k) + A - B\sin(\theta(k))$	$\theta(k) \in [0,2\pi]$ $0 < A < 2\pi;\ B > 0$
Bernoulli	$x(k+1) = \begin{cases} Bx(k) + A & x(k)<0 \\ f_{\mathrm{ber}}(x(k);A,B) & \\ Bx(k) - A & x(k)>0 \end{cases}$	$x(k) \in [-A,A]$ $0 < B < 2$
Tent	$x(k+1) = f_{\mathrm{tnt}}(x(k);A,B) = A - B\lvert x(k)\rvert$	$x(k) \in [A(1-B),A]$ $0 < B < 2$
Congruent	$x(k+1) = \begin{cases} Bx(k)-C & x(k)>A \\ f_{\mathrm{mod}}(x(k);A,B)=Bx(k) & \lvert x(k)\rvert \le A \\ Bx(k)+C & x(k)< -A \end{cases}$	$x(k) \in [-C,C]$ $1 < B < 2$ $C = 2A$
Hopping	$x(k+1) = \begin{cases} D(x(k)-A)+C & x(k)>A \\ f_{\mathrm{hop}}(x(k);A,B,D) = Bx(k) & \lvert x(k)\rvert \le A \\ D(x(k)+A)-C & x(k)<-A \end{cases}$	$x(k) \in [-C,C]$ $B, -D > 1$ $C = BA$ $f_{\mathrm{hop}}(-C) \in (0,C)$ $f_{\mathrm{hop}}(C) \in (-C,0)$
Henon	$x(k+1) = C + y(k) - Ax(k)^2$ $y(k+1) = Bx(k)$	$x(k), y(k)/B \in [-2/A,2/A]$ $A > (1-B)^2/4C,\ \lvert B\rvert < 1$ $3/4 < AC < 2$
Standard	$\rho(k+1) = \theta(k) + \rho(k)$ $\theta(k+1) = \theta(k) - B\sin(\rho(k))$	$\rho(k),\ \theta(k) \in [0,2\pi]$ $B > 0$
Lozi	$x(k+1) = C + y(k) - A\lvert x(k)\rvert$ $y(k+1) = Bx(k)$	$0 < B < 1$ $B + 1 < A < 2 - B/2$ $C > 1/2$
Arnold	$x(k+1) = f_{\mathrm{mod}}(x(k)+y(k);A,1)$ $y(k+1) = f_{\mathrm{mod}}(x(k)+By(k);A,1)$	$x(k), y(k) \in [-C,C]$ $1 < B < 2$ $C = 2A$
Baker	$x(k+1) = f_{\mathrm{mod}}(x(k);A,B)$ $y(k+1) = \begin{cases} By(k)+A & x(k)<0 \\ By(k)-A & x(k)>0 \end{cases}$	$x(k) \in [-C,C]$ $y(k) \in [-A,A]$ $1 < B < 2$ $C = 2A$

APPENDIX D

The C codes attached run the encryption-decryption program on the user's terminal. The audio capture and playback files are generated based on the sample codes provided in the Internet and the MSDN library-April 2003.

Aud_Stream1Dlg.cpp- the behavior of your application's main dialog.

Aud_Strm_DSnd_Cap.cpp - to create the capture sound buffer

Aud_Strm_DSnd_Play.cpp- to create the playback sound buffer

Aud_Strm_Encrypt.cpp -contains the two's compliment overflow nonlinearity encoder-decoder and bit swapping.

Aud_Strm_Net_Send.cpp and Aud_Strm_Net_Recv.cpp - to create the socket and send/receive TCP voice packets.

Aud_param.h - contains the user definable parameters i.e. sampling frequency, bits per sample and mono or stereo channel.

//Aud_Stream1.cpp

```
// Aud_Stream1.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Aud_Stream1.h"
#include "Aud_Stream1Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAud_Stream1App

BEGIN_MESSAGE_MAP(CAud_Stream1App, CWinApp)
	//{{AFX_MSG_MAP(CAud_Stream1App)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAud_Stream1App construction

CAud_Stream1App::CAud_Stream1App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CAud_Stream1App object

CAud_Stream1App theApp;

/////////////////////////////////////////////////////////////////////////////
// CAud_Stream1App initialization

BOOL CAud_Stream1App::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
```

```
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

        CAud_Stream1Dlg dlg;
        m_pMainWnd = &dlg;
        int nResponse = dlg.DoModal();
        if (nResponse == IDOK)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with OK
        }
        else if (nResponse == IDCANCEL)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with Cancel
        }

        // Since the dialog has been closed, return FALSE so that we exit the
        //  application, rather than start the application's message pump.
        return FALSE;
}
```

// Aud_Stream1Dlg.cpp

```
#include "stdafx.h"
#include <process.h>
#include "Aud_Stream1.h"
#include "Aud_Stream1Dlg.h"
#include "Aud_Strm_DSnd_Cap.h"
#include "Aud_Strm_DSnd_Play.h"
#include "Aud_Strm_Net_Send.h"
#include "Aud_Strm_Net_Recv.h"
#include "Aud_Strm_Encrypt.h"

#include "windowsx.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


Aud_Enc_Buff Aud_Enc_Buff_Fifo[NUM_PACKETS];
Aud_DS_Cap_Params Aud_Params;
Aud_Enc_Params Enc_Params;
static started_from_main;

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAud_Stream1Dlg dialog

CAud_Stream1Dlg::CAud_Stream1Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CAud_Stream1Dlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CAud_Stream1Dlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CAud_Stream1Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAud_Stream1Dlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAud_Stream1Dlg, CDialog)
	//{{AFX_MSG_MAP(CAud_Stream1Dlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_REC_MIC, OnRecMic)
	ON_BN_CLICKED(IDC_PLAY, OnPlay)
	ON_BN_CLICKED(IDC_STOP, OnStop)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAud_Stream1Dlg message handlers

BOOL CAud_Stream1Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	started_from_main= 0;

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CAud_Stream1Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CAud_Stream1Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CAud_Stream1Dlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CAud_Stream1Dlg::OnRecMic()
```

```
{
        // TODO: Add your control notification handler code here
        int loop;

        for(loop=0; loop < NUM_PACKETS; ++loop)
        {
                memset(&Aud_Enc_Buff_Fifo[loop].sample_bits, 0,PACKET_SIZE);
                Aud_Enc_Buff_Fifo[loop].filled = 0;
        }

        Aud_Params.frequency    = WAV_FREQ;
        Aud_Params.bits_sample  = WAV_BPS;
        Aud_Params.channel      = WAV_CHNL;
        Aud_Params.fifo_ptr     = Aud_Enc_Buff_Fifo;
        Aud_Params.g_hDlg       = GetSafeHwnd();
        Aud_Params.done                 = 0;

        Enc_Params.global_fifo = Aud_Enc_Buff_Fifo;
        Enc_Params.max_element_in_fifo = NUM_PACKETS;
        Enc_Params.size_of_element = PACKET_SIZE;
        Enc_Params.done = 0;


        Key_Gen_Encrypt_Decrypt();
        //Aud_Strm_Bit_Swap_Encrypt_Init();
        _beginthread((void(__cdecl*)(void*))Aud_Strm_Encrypt_Fifo_Thread, 0, (void *)
&Enc_Params);
        _beginthread((void(__cdecl*)(void*))Aud_Strm_DS_Capture, 0, (void *) &Aud_Params);

        started_from_main = 1;

        GetDlgItem(IDC_REC_MIC)->EnableWindow(FALSE);
        GetDlgItem(IDC_PLAY)->EnableWindow(FALSE);
        GetDlgItem(IDC_STOP)->EnableWindow(TRUE);
}

void CAud_Stream1Dlg::OnPlay()
{
        // TODO: Add your control notification handler code here
        int loop;

        for(loop=0; loop < NUM_PACKETS; ++loop)
        {
                memset(&Aud_Enc_Buff_Fifo[loop].sample_bits, 0,PACKET_SIZE);
                Aud_Enc_Buff_Fifo[loop].filled = 0;
        }

        Aud_Params.frequency    = WAV_FREQ;
        Aud_Params.bits_sample  = WAV_BPS;
        Aud_Params.channel      = WAV_CHNL;
        Aud_Params.fifo_ptr     = Aud_Enc_Buff_Fifo;
        Aud_Params.g_hDlg       = GetSafeHwnd();
        Aud_Params.done                 = 0;

        Enc_Params.global_fifo = Aud_Enc_Buff_Fifo;
        Enc_Params.max_element_in_fifo = NUM_PACKETS;
        Enc_Params.size_of_element = PACKET_SIZE;
        Enc_Params.done = 0;

        //Key_Gen_Encrypt_Decrypt();
        Aud_Strm_Bit_Swap_Decrypt_Init();
        _beginthread((void(__cdecl*)(void*))Aud_Strm_Decrypt_Fifo_Thread, 0, (void *)
&Enc_Params);
        _beginthread((void(__cdecl*)(void*))Aud_Strm_DS_Play, 0, (void *) &Aud_Params);

        started_from_main = 1;

        GetDlgItem(IDC_REC_MIC)->EnableWindow(FALSE);
        GetDlgItem(IDC_PLAY)->EnableWindow(FALSE);
        GetDlgItem(IDC_STOP)->EnableWindow(TRUE);
        GetDlgItem(IDC_REC_MIC)->EnableWindow(FALSE);
        GetDlgItem(IDC_PLAY)->EnableWindow(FALSE);
        GetDlgItem(IDC_STOP)->EnableWindow(TRUE);
}

void CAud_Stream1Dlg::OnStop()
{

        Aud_Params.done = 1;
        Enc_Params.done = 1;

        // TODO: Add your control notification handler code here
        GetDlgItem(IDC_REC_MIC)->EnableWindow(TRUE);
        GetDlgItem(IDC_PLAY)->EnableWindow(TRUE);
        GetDlgItem(IDC_STOP)->EnableWindow(FALSE);

}

void CAud_Stream1Dlg::OnOK()
{
        // TODO: Add extra validation here

        if(started_from_main)
        {
                while(Aud_Params.done!=3 && Enc_Params.done !=3)
                {
                }
        }

        CDialog::OnOK();
}

//-----------------------------------------------------------------------------
// Name: DspMsg(NULL, NULL, x,y,...)
// Desc: Displayes debug messages on the main screen
//-----------------------------------------------------------------------------
int _cdecl DspMsg( HWND hWnd,  HDC hDc,  int x,  int y,  LPSTR msgfmt, ... )
{
        int   igdc;
        LPSTR pcl,txbf;
        char  msgbf[768];
        void  FAR *VarArgList = (LPSTR *)&msgfmt + 1;
        HFONT hfnt,ofnt,hobj;

    wvsprintf( msgbf,msgfmt,(char *)VarArgList );
    if ( hDc )
                igdc = 0;
    else
        {
        hDc  = GetDC( hWnd );
        hfnt = (HFONT)SendMessage( hWnd,WM_GETFONT,0,0L );
        ofnt = (HFONT)SelectObject( hDc,hfnt );
        igdc = 1;
        }
    txbf = msgbf;
    hobj = (struct HFONT__ *)SelectObject( hDc,GetStockObject(SYSTEM_FIXED_FONT) );
    while ( *txbf )
        {
        pcl = _fstrchr( txbf,'\n' );
        if ( pcl )  *pcl++ = 0;
        TextOut( hDc,x,y,txbf,lstrlen(txbf) );
        if ( pcl )  txbf = pcl;
        else        break;
        y += 17;
        }
    SelectObject( hDc,hobj );
    if ( igdc )
        {
        SelectObject( hDc,ofnt );
        ReleaseDC( hWnd,hDc );
        }
    return ( y );
}
```

// Aud_Strm_DSnd_Cap.cpp

```
#include "stdafx.h"
#include "Aud_Strm_DSnd_Cap.h"
#include <windows.h>
#include <mmsystem.h> // used for multimedia wave format structure - audio mixer in this
case
#include <objbase.h>          // used for direct sound - because it's COM
#include <mmreg.h>            // needed for Windows Multimedia API
#include <dsound.h>           // direct sound header file
#include <process.h>          // needed for multithreading
//#include "windowsx.h"


static LPDIRECTSOUNDCAPTURE           g_pDSCapture           = NULL;
static LPDIRECTSOUNDCAPTUREBUFFER     g_pDSBCapture          = NULL;
static LPDIRECTSOUNDNOTIFY            g_pDSNotify            = NULL;
static DSBPOSITIONNOTIFY                     g_aCapPosNotify[ NUM_PACKETS + 1 ];
static HANDLE                         g_hCaptureNotificationEvents[NUM_PACKETS + 1];
static WAVEFORMATEX wfcapcap;


int _cdecl DspMsg( HWND hWnd,  HDC hDc,  int x,  int y,  LPSTR msgfmt, ... );

//-----------------------------------------------------------------------------
// Name: InitDirectSoundCapture()
// Desc: Initilizes DirectSound for capture
//-----------------------------------------------------------------------------
HRESULT InitDirectSoundCapture( HWND hDlg )
{
    HRESULT hr;

        // Free the memory for DirectSound notification structure
    ZeroMemory( &g_aCapPosNotify, sizeof(DSBPOSITIONNOTIFY) *
                (NUM_PACKETS + 1) );

    // Initialize COM
        hr = CoInitialize(NULL);

        if (FAILED(hr))
                return hr ;

        //create the DirectSoundCapture object
    if( FAILED( hr = DirectSoundCaptureCreate( NULL, &g_pDSCapture, NULL ) ) )
        return hr;

    return S_OK;
}


//-----------------------------------------------------------------------------
// Name: InitCaptureNotifications()
// Desc: Sets the notifications on the capture buffer which are handled
//       in WinMain1(). Sets two notifications - at the beginning and at the end.
//-----------------------------------------------------------------------------
HRESULT InitCaptureNotifications()
{
    HRESULT hr;

    if( NULL == g_pDSBCapture )
        return E_FAIL;

        // Create the notification interface
    if( FAILED( hr = g_pDSBCapture->QueryInterface( IID_IDirectSoundNotify,
                (VOID**)&g_pDSNotify ) ) )
        return hr;
```

```
     // Setup the notification positions
         for( int i = 0; i < NUM_PACKETS; i++ )
     {
         g_aCapPosNotify[i].dwOffset = ((PACKET_SIZE * i) + PACKET_SIZE) % (NUM_PACKETS *
PACKET_SIZE);
         g_aCapPosNotify[i].hEventNotify = g_hCaptureNotificationEvents[i];
     }

     if( FAILED( hr = g_pDSNotify->SetNotificationPositions(NUM_PACKETS,
g_aCapPosNotify)))
                 return hr;

     return S_OK;
}



//-----------------------------------------------------------------------------
// Name: CreateCaptureBuffer()
// Desc: Sets the format, creates a capture buffer, set notifications for
//          DirectSound Capture and initialize the network for multicasting.
//-----------------------------------------------------------------------------
HRESULT CreateCaptureBuffer(int freq, int bits_per_sample, int channels)
{
    HRESULT hr;        // keep error message if any
    DSCBUFFERDESC dscbd;// Create a DirectSound buffer description structure
        WAVEFORMATEX wfcap; // Create a wave format structure

        // Release and free the memory for the wave format structure
        ZeroMemory (&wfcap, sizeof (WAVEFORMATEX));

        // Release and free the memory for the DirectSound capture and notify structure
        SAFE_FREE( g_pDSNotify );
        SAFE_FREE( g_pDSBCapture );

        // Sets the wave format
        wfcap.wFormatTag                = WAVE_FORMAT_PCM;
        wfcap.nChannels                 = channels;
        wfcap.nSamplesPerSec   = freq; //11025;
        wfcap.wBitsPerSample   = bits_per_sample;//16

        wfcap.nBlockAlign               = wfcap.nChannels * wfcap.wBitsPerSample / 8;
        wfcap.nAvgBytesPerSec  = wfcap.nBlockAlign * wfcap.nSamplesPerSec;

        SAFE_FREE( g_pDSNotify );
        SAFE_FREE( g_pDSBCapture );

        // Sets the capture buffer description
        ZeroMemory( &dscbd, sizeof(dscbd) );
        dscbd.dwSize          = sizeof(dscbd);
        dscbd.dwFlags              = DSCBCAPS_WAVEMAPPED ;
        dscbd.dwBufferBytes = 1 * wfcap.nAvgBytesPerSec;//CAP_BUFFER_SIZE ; //5512 ; //1 *
wfcap.nAvgBytesPerSec;
        dscbd.lpwfxFormat    = (LPWAVEFORMATEX) &wfcap; // Set the format during creation

        // Create the DirectSound capture buffer
            if( FAILED( hr = g_pDSCapture->CreateCaptureBuffer( &dscbd,
                &g_pDSBCapture, NULL ) ) )
                return hr;

        // Set notifications for DirectSound Capture
          if( FAILED( InitCaptureNotifications() ) )
         return hr;

         return S_OK;
        }

//-----------------------------------------------------------------------------
// Name: FreeDirectSoundCapture()
// Desc: Releases DirectSound
//-----------------------------------------------------------------------------
```

```
HRESULT FreeDirectSoundCapture()
{
        // Release DirectSoundCapture interfaces for capturing
    SAFE_FREE( g_pDSNotify );
    SAFE_FREE( g_pDSBCapture );
    SAFE_FREE( g_pDSCapture );

    // Release COM
    CoUninitialize();

    return S_OK;
}

//-----------------------------------------------------------------------------
// Name: Aud_Strm_DS_Capture()
// Desc: threaded function that captures wave data from mic via DirectSound
//-----------------------------------------------------------------------------
void __cdecl Aud_Strm_DS_Capture(void * params)
{
        int loop1;
        void * ptr_to_ds_data;
        int ds_data_size;
        int index;
        HRESULT hr;
        int max_element_in_fifo = 0;

        Aud_DS_Cap_Params_ptr aud_params;
        unsigned int  frequency;
        unsigned char bits_sample;
        unsigned char channel;
        Aud_Enc_Buff_ptr global_fifo;
        unsigned int * done;
        HWND g_hDlg;
        unsigned int fifo_index;


        if(!params)
        {
                AfxMessageBox("DS CAPT NO PARAMS!", MB_OK, NULL);
                _endthread();
        }

        aud_params = (Aud_DS_Cap_Params_ptr) params;
        frequency = aud_params->frequency;
        bits_sample = aud_params->bits_sample;
        channel = aud_params->channel;
        global_fifo = aud_params->fifo_ptr;
        done = &(aud_params->done);
        g_hDlg = aud_params->g_hDlg;
        max_element_in_fifo = NUM_PACKETS;

    SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_ABOVE_NORMAL);

        //Get the handle to the foreground window
        if (g_hDlg == NULL)
        {
                g_hDlg = GetDesktopWindow();
        }

        //create some event handles used later for DS capture notifcations
        for (loop1 = 0; loop1<NUM_PACKETS ; ++loop1)
         {
                 g_hCaptureNotificationEvents[loop1] = CreateEvent(NULL, FALSE, FALSE,
NULL);
                 if (NULL == g_hCaptureNotificationEvents[loop1])
                {
                        MessageBox(NULL, "Failed on CreateEvent for audio capture",
"Aud_Cap" , MB_OK);
                        *done = 3;
                        _endthread();
                }
    }

    // Initialize DirectSound
    if( FAILED( InitDirectSoundCapture( g_hDlg ) ) )
    {
        MessageBox(NULL, "Error Initializing DirectSound Capture", "Aud_Cap", MB_OK |
MB_ICONERROR);
                FreeDirectSoundCapture();
                *done = 3;
                _endthread();
    }

        // Create the sound capture buffer
        if( FAILED( CreateCaptureBuffer(frequency, bits_sample, channel) ) )
    {
        MessageBox(NULL, "Error Creating DirectSoundCapture Buffer.","Aud_Cap", MB_OK |
MB_ICONERROR );
        FreeDirectSoundCapture();
        *done = 3;
        _endthread();
    }

// Start recording sound from mic
// Tell the capture buffer to start recording

    if( FAILED( hr = g_pDSBCapture->Start( DSCBSTART_LOOPING ) ) )
        {
                MessageBox(NULL, "Error Starting DirectSoundCapture.","Aud_Cap", MB_OK |
MB_ICONERROR );
                FreeDirectSoundCapture();
                *done = 3;
                _endthread();
        }

        fifo_index = 0;
        // This is the loop for processing the event which is capturing audio.
        while( !*done )
         {
            index = MsgWaitForMultipleObjects( NUM_PACKETS, g_hCaptureNotificationEvents,
                                               FALSE, 1000, QS_ALLEVENTS);

            // This means that DirectSound just finished filling
           // a section of the buffer, so we need to fill the
          // global Aud_Enc_Buff buffer with new raw wav data


                index -= WAIT_OBJECT_0;
                if(index<NUM_PACKETS && index>=0)
         {
                if(!FAILED( hr = g_pDSBCapture->Lock( PACKET_SIZE * index,

                PACKET_SIZE,

                (LPVOID *) &ptr_to_ds_data,

                (unsigned long*)&ds_data_size,

                 NULL, 0, 0L)))
                        {
                                if(fifo_index == max_element_in_fifo)
                                        fifo_index = 0;

                                while(global_fifo[fifo_index].filled)
                                        Sleep(2);
                        }

                        {
                                memcpy((unsigned char *)global_fifo[index].sample_bits,
ptr_to_ds_data, ds_data_size);
```

```
                                DspMsg (0,0,100,100,"copied from mic index = %d size =
%d",index,ds_data_size);

                                // Unlock the capture buffer
                                g_pDSBCapture->Unlock( ptr_to_ds_data, ds_data_size, NULL,
0 );

                                global_fifo[index].filled = 1;
                        }

                }
                else break;
        }

   // Stop the recording
    g_pDSBCapture->Stop();

    // Clean up everything
    SAFE_FREE( g_pDSNotify );
    SAFE_FREE( g_pDSBCapture );
    SAFE_FREE( g_pDSCapture );

    // Release COM
    CoUninitialize();


        for (loop1 = 0; loop1< NUM_PACKETS; ++loop1)
        {
                CloseHandle( g_hCaptureNotificationEvents[loop1] );
        }

        *done = 3;

        _endthread();

}
```

//Aud_Strm_DSnd_Play.cpp

```
#include "stdafx.h"
#include "Aud_Strm_DSnd_Play.h"
#include "Aud_Strm_DSnd_Cap.h"
#include <windows.h>
#include <mmsystem.h>// used for multimedia wave format structure - audio mixer in this
case
#include <objbase.h>// used for direct sound - because it's COM
#include <mmreg.h> // needed for Windows Multimedia API
#include <dsound.h> // direct sound header file
#include <process.h> // needed for multithreading
//#include "windowsx.h"

LPDIRECTSOUND                 g_pDS                   = NULL;
LPDIRECTSOUNDBUFFER           pDSBPrimary             = NULL;
LPDIRECTSOUNDBUFFER                   g_pDSPlayBuffer                 = NULL;
LPDIRECTSOUNDNOTIFY           g_pDSPlayNotify                 = NULL;
DSBPOSITIONNOTIFY             g_aPlayPosNotify[NUM_PACKETS + 1];
HANDLE                        g_hPlayNotificationEvents[NUM_PACKETS + 1];
static WAVEFORMATEX            wfplay;

int _cdecl DspMsg( HWND hWnd,  HDC hDc,  int x,  int y,  LPSTR msgfmt, ... );


//-----------------------------------------------------------------------------
// Name: InitDirectSoundPlayback()
// Desc: Initilizes DirectSound for playback
//-----------------------------------------------------------------------------
HRESULT InitDirectSoundPlayback( HWND hDlg, int freq, int bits_per_sample, int channels)
{
    HRESULT hr;
    // Initialize COM
       hr = CoInitialize(NULL);

       if (FAILED(hr))
              return hr ;

// Create IDirectSound using the primary sound device
    if( FAILED( hr = DirectSoundCreate( NULL, &g_pDS, NULL ) ) )
        return hr;

// Set coop level to DSSCL_PRIORITY so that wave volume can be set
    if( FAILED( hr = g_pDS->SetCooperativeLevel( hDlg, DSSCL_PRIORITY )))
        return hr;

 // Get the primary buffer
    DSBUFFERDESC        dsbd;
    ZeroMemory( &dsbd, sizeof(DSBUFFERDESC) );
    dsbd.dwSize        = sizeof(DSBUFFERDESC);
    dsbd.dwFlags       = DSBCAPS_PRIMARYBUFFER | DSBCAPS_CTRLVOLUME;
    dsbd.dwBufferBytes = 0;
    dsbd.lpwfxFormat   = NULL;

// Create a primary audio playback buffer
    if( FAILED( hr = g_pDS->CreateSoundBuffer( &dsbd, &pDSBPrimary, NULL ) ) )
        return hr;

// Set primary buffer format to 44kHz, 8-bit and mono output.
    ZeroMemory( &wfplay, sizeof(WAVEFORMATEX) );
    wfplay.wFormatTag      = WAVE_FORMAT_PCM;
    wfplay.nChannels       = channels;
    wfplay.nSamplesPerSec  = freq;
    wfplay.wBitsPerSample  = bits_per_sample;
    wfplay.nBlockAlign     = wfplay.wBitsPerSample / 8 * wfplay.nChannels;
    wfplay.nAvgBytesPerSec = wfplay.nSamplesPerSec * wfplay.nBlockAlign;

    if( FAILED( hr = pDSBPrimary->SetFormat(&wfplay) ) )
        return hr;

       // Free up the primary audio buffer
    SAFE_FREE( pDSBPrimary );

    return S_OK;
}


//-----------------------------------------------------------------------------
// Name: FreeDirectSoundPlay()
// Desc: Releases DirectSound
//-----------------------------------------------------------------------------
HRESULT FreeDirectSoundPlay(void)
{
       // Release DirectSoundCapture interfaces for listening
       SAFE_FREE(g_pDSPlayNotify);

       if (g_pDSPlayBuffer != NULL)
                     SAFE_FREE( g_pDSPlayBuffer );

    SAFE_FREE( g_pDS );

    // Release COM
    CoUninitialize();

    return S_OK;
}

//-----------------------------------------------------------------------------
// Name: CreateStreamingPlaybackBuffer()
// Desc: Creates a streaming buffer, and the notification events to handle
//       filling it as sound is played
//-----------------------------------------------------------------------------
```

```
HRESULT CreateStreamingPlaybackBuffer(void)
{
    HRESULT hr;
        DSBUFFERDESC dsbd;
        char*   _pbBuffer;
        DWORD   dwBufferLength;

        // Release and free the memory for the notification structure
         ZeroMemory( &g_aPlayPosNotify, sizeof(DSBPOSITIONNOTIFY) *
                (NUM_PACKETS +1) );//start to comment here

// Sets the playback buffer description
// Set up the direct sound buffer, and only request the flags needed
// since each requires some overhead and limits if the buffer can
// be hardware accelerated
    ZeroMemory( &dsbd, sizeof(DSBUFFERDESC) );
    dsbd.dwSize         = sizeof(DSBUFFERDESC);
    dsbd.dwFlags        =       DSBCAPS_CTRLPOSITIONNOTIFY | // Needed for notification
                DSBCAPS_GETCURRENTPOSITION2 |
                DSBCAPS_GLOBALFOCUS;

dsbd.dwBufferBytes = 1 * wfplay.nAvgBytesPerSec;
dsbd.lpwfxFormat    = (LPWAVEFORMATEX) &wfplay;

    // Create a DirectSound buffer
        if( FAILED( hr = g_pDS->CreateSoundBuffer( &dsbd, &g_pDSPlayBuffer, NULL )))
                return hr;


        if( FAILED( hr = g_pDSPlayBuffer->Lock(       0,
                                        NULL,
                                        (LPVOID *)&_pbBuffer,
                                        &dwBufferLength,
                                        NULL,
                                        0,
                                        DSBLOCK_ENTIREBUFFER ) ) )
        {
                MessageBox(NULL,"Lock EntireBuffer failed", "Failed", MB_OK);
        }

        else
        {
                // Fill with silence
                memset(_pbBuffer, 0x7f, dwBufferLength);
        }

        // Unlock the playback buffer
        g_pDSPlayBuffer->Unlock(_pbBuffer, dwBufferLength, NULL, 0);

        return S_OK;
}


//-----------------------------------------------------------------------------
// Name: InitPlaybackNotifications()
// Desc: sets up the direct sound notification by setting
//       the offset trigger and passing the event handle
//-----------------------------------------------------------------------------
HRESULT InitPlaybackNotifications(void)
{
        int loop1;
        HRESULT hr;

        for (loop1 = 0; loop1<NUM_PACKETS ; ++loop1)
         {
                g_aPlayPosNotify[loop1].dwOffset = PACKET_SIZE * loop1;
                g_aPlayPosNotify[loop1].hEventNotify = g_hPlayNotificationEvents[loop1];
        }

           if( FAILED(g_pDSPlayBuffer-
>QueryInterface(IID_IDirectSoundNotify,(void**)&g_pDSPlayNotify)))
```

```
        {
                MessageBox(NULL,"Failed on IDirectSoundCaptureBuffer_QueryInterface",
"Debug" , MB_OK);
                        return -1;
        }

    hr = g_pDSPlayNotify->SetNotificationPositions(NUM_PACKETS, g_aPlayPosNotify);
        if(FAILED(hr))
    {
        switch (hr)
                {
                        case DSERR_INVALIDPARAM:
                                break;
                        case DSERR_OUTOFMEMORY:
                                break;
                        default :
                                break;
                }
                return hr;
    }

        return S_OK;
}

//-----------------------------------------------------------------------------
// Name: Aud_Strm_DS_Play()
// Desc: threaded function that plays
//       back wave data from global buffer via DirectSound
//-----------------------------------------------------------------------------
void __cdecl Aud_Strm_DS_Play(void * params)
{

        int loop1;
        //void * ptr_to_ds_data;
        //int ds_data_size;
        int play_index;
        int fifo_index;
        HRESULT hr;

        Aud_DS_Cap_Params_ptr aud_params;
        unsigned int  frequency;
        unsigned char bits_sample;
        unsigned char channel;
        Aud_Enc_Buff_ptr global_fifo;
        unsigned int * done;
        HWND g_hDlg;

    void * pbInput1 = NULL;
    void * pbInput2 = NULL;
        unsigned long szInput1;
        unsigned long szInput2;
        unsigned long status;

        if(!params)
        {
                AfxMessageBox("DS PLAY NO PARAMS!", MB_OK, NULL);
                _endthread();
        }

        aud_params = (Aud_DS_Cap_Params_ptr) params;
        frequency = aud_params->frequency;
        bits_sample = aud_params->bits_sample;
        channel = aud_params->channel;
        global_fifo = aud_params->fifo_ptr;
        done = &(aud_params->done);
        g_hDlg = aud_params->g_hDlg;

        SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_ABOVE_NORMAL);

        //Get the handle to the foreground window
        if (g_hDlg == NULL)
        {
                g_hDlg = GetDesktopWindow();
        }


        //create some event handles used later for DS playback notifcations
        for (loop1 = 0; loop1< NUM_PACKETS; ++loop1)
    {
        g_hPlayNotificationEvents[loop1] = CreateEvent(NULL, FALSE, FALSE, NULL);
        if (NULL == g_hPlayNotificationEvents[loop1])
                {
                        MessageBox(NULL,"Failed on CreateEvent", "Debug" , MB_OK);
                        *done = 3;
                        _endthread();
                }
    }

        // Init DirectSound
    if( FAILED(InitDirectSoundPlayback(g_hDlg, frequency, bits_sample, channel) ) )
    {
                MessageBox(NULL, "Error Initializing DirectSound Playback", "Aud_Play" ,
MB_OK);
                // Clean up everything
                FreeDirectSoundPlay();
                *done = 3;
                _endthread();
    }

        // Create the sound buffer object from the  data
        if( FAILED(CreateStreamingPlaybackBuffer() ) )
        {
                MessageBox( g_hDlg, "Error Creating DirectSound Playback Buffer.",
"Aud_Play", MB_OK);
                // Clean up everything
                FreeDirectSoundPlay();
                *done = 3;
                _endthread();
        }

        // Init Notifications
    if( FAILED(InitPlaybackNotifications() ) )
    {
                MessageBox(NULL, "Error Setting-up Playback Notifications", "Aud_Play" ,
MB_OK);
                // Clean up everything
                FreeDirectSoundPlay();
                *done = 3;
                _endthread();
    }


        g_pDSPlayBuffer->Play(0, 0, DSBPLAY_LOOPING);
        DspMsg (0,0,200,250,"playback start   ");

        play_index = 0;
        fifo_index = 0;
        while( !*done )
    {
                DspMsg (0,0,200,250,"playback start  1");

                if(fifo_index == NUM_PACKETS)
                        fifo_index = 0;

                DspMsg (0,0,200,250,"playback start  2");

                if(!(global_fifo[fifo_index].filled))
                {
                        Sleep(1);
                        continue;
                }
```

```
            DspMsg (0,0,200,250,"playback start  3");

            g_pDSPlayBuffer->GetStatus(&status);
            if( status & DSBSTATUS_BUFFERLOST )
            {
                    g_pDSPlayBuffer->Restore();
                    g_pDSPlayBuffer->Play( 0, 0, DSBPLAY_LOOPING );
            }

            DspMsg (0,0,200,250,"playback start  4");

            play_index = MsgWaitForMultipleObjects(NUM_PACKETS,         // How many
possible events

            g_hPlayNotificationEvents,// Location of handles
             FALSE,                              // Wait for all?
             1000,                               // How long to wait
             QS_ALLEVENTS);                              // Any message is an event

            DspMsg (0,0,200,250,"playback start  5");

            play_index -= WAIT_OBJECT_0;


            if(play_index < NUM_PACKETS)
            {
                    play_index+=6;

                    if (play_index > (NUM_PACKETS - 1))
                            play_index -= NUM_PACKETS;

                    hr= g_pDSPlayBuffer->Lock(    PACKET_SIZE * play_index,
                                            PACKET_SIZE,
                                            &pbInput1,
                                            &szInput1,
                                            &pbInput2,
                                            &szInput2,
                                            0);

                    if(FAILED(hr))
                    {
                            MessageBox(NULL,"Playback Buffer Lock failed", "Aud_Play",
MB_OK);
                            global_fifo[fifo_index].filled=0;
                            ++fifo_index;
                            DspMsg (0,0,200,280,"playback start  F");
                            continue;
                    }
                    else
                    {
                            memcpy(pbInput1, global_fifo[fifo_index].sample_bits,
szInput1);
                            g_pDSPlayBuffer->Unlock(pbInput1, szInput1, pbInput2,
szInput2);
                            global_fifo[fifo_index].filled = 0;
                            DspMsg (0,0,300,400,"play to direct sound = %d size =
%d",fifo_index,szInput1);
                            ++fifo_index;
                            DspMsg (0,0,200,250,"playback start  6");
                    }

            }
        }//while !*done

        DspMsg (0,0,200,250,"playback stopped");

        //Stop audio playback
        g_pDSPlayBuffer->Stop();

        for (loop1 = 0; loop1<NUM_PACKETS; ++loop1)
        {
                CloseHandle( g_hPlayNotificationEvents[loop1]);
        }

    // Clean up everything
    FreeDirectSoundPlay();

        *done = 3;

        _endthread();
}
```

//Aud_Strm_Encrypt.cpp

```
#include "stdafx.h"
#include "Aud_Strm_Encrypt.h"
#include "Aud_Strm_DSnd_Cap.h"
#include "Aud_Strm_Net_Recv.h"
#include "Aud_Strm_Net_Send.h"
#include "process.h"
#include "math.h"
#include "string.h"

#define DELAY 3

BOOLEAN b[32];

int ctr = 0;
long double x[3];
long double log_val[4];
double key[DELAY];
double max_norm=0.0;          //change values here
double max_s_val_new=0.0;
double min=1.0;
double mod(double x);

int _cdecl DspMsg( HWND hWnd,  HDC hDc,  int x,  int y,  LPSTR msgfmt, ... );

//TWO'S COMPLIMENT OVERFLOW NONLINEARITY

void Key_Gen_Encrypt_Decrypt(void)
{

        log_val[0]=INIT_CONDITION;

        for (int n=0;n<3;n++)
        {
                log_val[n+1]=3.6 * log_val[n] * (1-log_val[n]);
        //generating chaotic value x[i+1]
                key[n]=log_val[n+1];
        }
}


double mod(double x)
{
        return (x - 2 * floor((x + 1) / 2));
}


void Aud_Strm_Encrypt_Sample(unsigned char *input_val, unsigned char *output_val)
{
        double temp_val[PACKET_SIZE], intermediate_val[PACKET_SIZE];
        double enc_sig_buffer[PACKET_SIZE];
        double sum_val;
        double encoded_sig;
        int i, j;
        int k=0;

        for (i=0; i<PACKET_SIZE; i++)
        {
                if ((double)input_val[i]>max_norm)
                {
                        max_norm=(double)input_val[i];
                }
        }

        for (i=0; i<PACKET_SIZE; i++)
        {
                intermediate_val[i]=((double)input_val[i])/255.0;
        }

        for (i=0; i<PACKET_SIZE; i++)
        {
                enc_sig_buffer[i] = 0.0;

        }

        for (i=0; i<PACKET_SIZE; i++)
        {

                sum_val = 0.0;

                for (j=1; j<=DELAY; j++)
                {
                        sum_val = sum_val + enc_sig_buffer[j-1]*key[j-1];

                }

                temp_val[i]=mod(intermediate_val[i]+sum_val);


                for (k=DELAY; k>0; k--)
                {
                        encoded_sig = enc_sig_buffer[k-1];
                        enc_sig_buffer[k] = encoded_sig;
                }

                enc_sig_buffer[0] = temp_val[i];

        }


        for (i=0; i<PACKET_SIZE; i++)
        {
                intermediate_val[i] = temp_val[i];
                if (intermediate_val[i]<min)
                {
                        min = intermediate_val[i];
                }
        }

        for (i=0; i<PACKET_SIZE; i++)
        {
                intermediate_val[i]=intermediate_val[i]-min;
        }

        for (i=0; i<PACKET_SIZE; i++)
        {
                if (intermediate_val[i]>max_s_val_new)
                {
                        max_s_val_new=intermediate_val[i];
                }
        }


        for (i=0; i<PACKET_SIZE; i++)
        {

                output_val[i]=(unsigned char)(((intermediate_val[i])/max_s_val_new)*255);
        }
}
```

```
void Aud_Strm_Decrypt_Sample(unsigned char *input_val, unsigned char *output_val)
{
        double temp_val[PACKET_SIZE],
intermediate_val[PACKET_SIZE];//,intermediate_temp[PACKET_SIZE];;
        double enc_sig_buffer[PACKET_SIZE];
        double encoded_signal;
        double sum_val;
        int i, j;
        int k=0;


        for (i=0; i<PACKET_SIZE; i++)
        {
                intermediate_val[i]=(((((double)input_val[i]))*2.0)/255);

        }


        for (i=0; i<PACKET_SIZE; i++)
        {
                intermediate_val[i]=intermediate_val[i]+(-1.0);

        }

        for (i=0; i<PACKET_SIZE; i++)
        {
                enc_sig_buffer[i] = 0.0;
        }

        for (i=0; i<PACKET_SIZE; i++)
        {

                sum_val = 0.0;

                for (j=1; j<=DELAY; j++)
                {
                        sum_val = sum_val + key[j-1]*enc_sig_buffer[j-1];
                }

                temp_val[i]=mod(intermediate_val[i]-sum_val);

                if (temp_val[i] < 0)
                {
                        temp_val[i] = temp_val[i] * (-1.0);
                }

                for (k=DELAY; k>0; k--)
                {
                        encoded_signal = enc_sig_buffer[k-1];
                        enc_sig_buffer[k] = encoded_signal;
                }

                enc_sig_buffer[0] = intermediate_val[i];
        }

        for (i=0; i<PACKET_SIZE; i++)
        {
                intermediate_val[i] = temp_val[i];
                output_val[i]=(unsigned char)((intermediate_val[i]*255.0) +0.5);

        }
}

//BIT SWAPPING

void Aud_Strm_Bit_Swap_Encrypt_Init(void)
{
        x[0] = INIT_CONDITION;
        x[1] = 0; /* dont matter coz will be derived later from map*/
        x[2] = 0; /* dont matter coz will be derived later from map*/
        ctr = 0;

        // next sample will be the first.
}


void Aud_Strm_Bit_Swap_Decrypt_Init(void)

{
        x[0] = INIT_CONDITION;
        x[1] = 0; /* dont matter coz will be derived later from map*/
        x[2] = 0; /* dont matter coz will be derived later from map*/
        ctr = 0;

        // next sample will be the first.
}

unsigned char Aud_Strm_Bit_Swap_Encrypt_Sample(unsigned char sample_byte)
{
        int i, j;
        double remainder;
        BOOLEAN temp_bool, sample_bit[8];
        unsigned char sample_new;

        if (ctr % 4 == 0)                                                   //checking for
every 4 in-coming samples
        {

                x[1]=3.9 * x[0] * (1-x[0]);                         //generating chaotic
value x[i+1]
                x[2]=3.9 * x[1] * (1-x[1]);                         //generating chaotic
value x[i+2]

                for (j=0;j<2;++j)
                {
                        remainder=x[j+1];

                        for (i=0;i<16;i++)
                        {
                                if ( (remainder == 0) || (remainder - pow(2, (-1*(i+1))) <
0) )
                                {
                                        b[(j*16)+i]=0;
                                }
                                else
                                {
                                        remainder=remainder - pow(2, (-1*(i+1)));
                                        b[(j*16)+i]=1;
                                }
                        }
                }

                x[0]=x[2];
                x[1]=0;
                x[2]=0;
        }

        for (i=0;i<8;i++)
        {
                sample_bit[i]=(sample_byte>>i)&1;
        }

        for (i=0;i<4;i++)
        {
                if (b[((ctr%4)*4)+i])
                {
                        temp_bool = sample_bit[i];
                        sample_bit[i] = sample_bit[i+4];
                        sample_bit[i+4] = temp_bool;
                }
```

```
    }
    for (i=1;i<=7;i+=2)
    {       sample_bit[i] = sample_bit[i] ^ b[((ctr%4)*4)+i];

    }
    sample_new=0;
    for (i=0;i<8;i++)
    {       sample_new= sample_new | (sample_bit[i]<<i);

    }
    ctr++;
    return sample_new;
}
unsigned char Aud_Strm_Bit_Swap_Decrypt_Sample(unsigned char sample_byte)
{       int i, j;
        double remainder;
        BOOLEAN temp_bool, sample_bit[8];                                   //checking for
        unsigned char sample_old;

        if (ctr % 4 == 0)
  every 4 in-coming samples                                             //generating chaotic
        {                                                               //generating chaotic
                x[1]=3.9 * x[0] * (1-x[0]);
  value x[i+1]
                x[2]=3.9 * x[1] * (1-x[1]);
  value x[i+2]
                for (j=0;j<2;++j)
                {       remainder=x[j+1];
                        for (i=0;i<16;i++)
                        {       if ( (remainder == 0) || (remainder - pow(2, (-1*(i+1))) <
  0) )
                                {       b[(j*16)+i]=0;

                                }
                                else
                                {       remainder=remainder - pow(2, (-1*(i+1)));
                                        b[(j*16)+i]=1;

                                }
                        }
                }
                x[0]=x[2];
                x[1]=0;
                x[2]=0;
        }
        for (i=0;i<8;i++)
        {       sample_bit[i]=(sample_byte>>i)&1;

        }
        for (i=1;i<=7;i+=2)
        {       sample_bit[i] = sample_bit[i] ^ b[((ctr%4)*4)+i];

        }
        for (i=0;i<4;i++)
        {       if (b[((ctr %4)*4)+i])
```

```
			{
				temp_bool = sample_bit[i];
				sample_bit[i] = sample_bit[i+4];
				sample_bit[i+4] = temp_bool;
			}
		}

		sample_old=0;

		for (i=0;i<8;i++)
		{
			sample_old= sample_old | (sample_bit[i]<<i);
		}

		ctr++;

		return sample_old;
}

void __cdecl Aud_Strm_Encrypt_Fifo_Thread(void * param)
{
		int index, loop, y=0;
		Aud_Enc_Buff_ptr global_fifo;
		int max_element_in_fifo;
		int size_of_element;
		unsigned int * done;
		Aud_Enc_Params_ptr aud_enc_param = (Aud_Enc_Params_ptr) param;
		unsigned char local_encrypted_data[PACKET_SIZE];
		unsigned char output_Aud_Strm_Encrypt_Sample[PACKET_SIZE];

		FILE * file;
		FILE * file1;

		global_fifo = aud_enc_param->global_fifo;
		max_element_in_fifo = aud_enc_param->max_element_in_fifo;
		size_of_element = aud_enc_param->size_of_element;
		done = &(aud_enc_param->done);

		SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_ABOVE_NORMAL);

		index = 0;

		Key_Gen_Encrypt_Decrypt();			//iniate key for first level
encryption

		Aud_Strm_Bit_Swap_Encrypt_Init();	//iniate key for second level encryption


	//CreateSocket for sending
		if(Aud_Strm_Net_Send_Init())
		{
			*done = 3;
			_endthread();
			return;
		}

		file = fopen("c:\\encrypted.wav","wb");
		file1 = fopen("c:\\raw.wav","wb");

		while(! *done)
		{
			if(index == max_element_in_fifo)
			    index = 0;

			if(!(global_fifo[index].filled))
			{
				Sleep(1);
				continue;
			}
```

```
        Aud_Strm_Encrypt_Sample(global_fifo[index].sample_bits,output_Aud_Strm_Encrypt_Sam
ple);

                global_fifo[index].filled = 0;

                for (loop=0;loop<PACKET_SIZE;loop++)
                {
                        local_encrypted_data[loop]
                        =Aud_Strm_Bit_Swap_Encrypt_Sample(output_Aud_Strm_Encrypt_Sample[lo
                        op]);

                }

                fwrite((const void *)local_encrypted_data, PACKET_SIZE, 1, file);
                fwrite((const void *)global_fifo[index].sample_bits, PACKET_SIZE, 1,
file1);

                DspMsg (0,0,100,120,"encrypt to index = %d size = %d",index,PACKET_SIZE);


                Aud_Strm_Net_Send(local_encrypted_data);

                ++index;
        }


        fclose(file);
        fclose(file1);


        *done = 3;

        //closesocket
        Aud_Strm_Net_Send_Clean();

        _endthread();

}


void __cdecl Aud_Strm_Decrypt_Fifo_Thread(void * param)
{
        int index, loop;
        Aud_Enc_Buff_ptr global_fifo;
        int max_element_in_fifo;
        int size_of_element;
        unsigned int * done;
        unsigned char local_to_decrypt[PACKET_SIZE];
//      unsigned char local_done[PACKET_SIZE];
        unsigned char local_to_be_decrypted[PACKET_SIZE];
        Aud_Enc_Params_ptr aud_enc_param = (Aud_Enc_Params_ptr) param;

        //FILE *file;
        FILE *file1;

        global_fifo = aud_enc_param->global_fifo;
        max_element_in_fifo = aud_enc_param->max_element_in_fifo;
        size_of_element = aud_enc_param->size_of_element;
        done = &(aud_enc_param->done);

        SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_ABOVE_NORMAL);

        index = 0;

        Aud_Strm_Bit_Swap_Decrypt_Init();

        Key_Gen_Encrypt_Decrypt();

        //create socket recv
        if(Aud_Strm_Net_Recv_Init())
        {
                *done = 3;
                _endthread();
                return;
        }

        //file = fopen("c:\\encrypted.wav","rb");
        file1 = fopen("c:\\decrypted.wav","wb");

        DspMsg (0,0,200,420,"reading start   ");


        while(! *done)
        {

                DspMsg (0,0,200,420,"reading start  1");

                if(index == max_element_in_fifo)
                        index = 0;

                DspMsg (0,0,200,420,"reading start  2");

                if((global_fifo[index].filled))
                {
                        Sleep(1);
                        continue;
                }

                DspMsg (0,0,200,420,"reading start  3 ");


                //recv socket directly into local_to_be_decrypted_data[index]
                if(Aud_Strm_Net_Recv((unsigned char *)local_to_be_decrypted))
                {
                        DspMsg (0,0,200,420,"reading start  3a");
                        break;
                }

                //if (feof(file))
                //      break;

                //fread((void *)local_to_be_decrypted,PACKET_SIZE, 1, file);

                DspMsg (0,0,200,420,"reading start  4 ");
                DspMsg (0,0,300,520,"decrypt to index = %d size = %d",index,PACKET_SIZE);



                for(loop=0;loop<PACKET_SIZE;loop++)
                {
                        local_to_decrypt[loop]=
Aud_Strm_Bit_Swap_Decrypt_Sample(local_to_be_decrypted[loop]);
                }

                Aud_Strm_Decrypt_Sample(local_to_decrypt,global_fifo[index].sample_bits);

        //Aud_Strm_Decrypt_Sample(local_to_be_decrypted,global_fifo[index].sample_bits);


                global_fifo[index].filled = 1;

                DspMsg (0,0,200,420,"reading start  5");

                fwrite((void *)global_fifo[index].sample_bits, PACKET_SIZE, 1, file1);

                ++index;
        }


        DspMsg (0,0,200,420,"reading ending ");
        fclose(file1);

        Aud_Strm_Net_Recv_Clean();

        *done = 3;

        _endthread();

}
```

//Aud_Strm_Net_Send.cpp

```
#include "stdafx.h"
#include "Aud_Strm_Net_Send.h"
#include "winsock.h"
#include "Aud_params.h"

static WORD             wVersionRequested;
static WSADATA          wsaData;
static SOCKET            sendsock;
static SOCKET                   sessionsock;
static char                      chLocalAddress[16];
static struct            sockaddr_in sock_addr;// new_sock_addr;
static struct           sockaddr_in send_addr;// target_sock_addr;
static DWORD            Err;
static u_short          usPort;
static int                      true_int;
static int                      iRet;

int _cdecl DspMsg( HWND hWnd,  HDC hDc,  int x,  int y,  LPSTR msgfmt, ... );
void Aud_Strm_Net_Err_Translate(int error_val);

int Aud_Strm_Net_Send_Init(void)
{
        int error;

        wVersionRequested = MAKEWORD(2,0);
        iRet = WSAStartup(wVersionRequested, &wsaData);//request winsock version + service
        if (iRet != 0)
        {
                MessageBox(NULL,"Failed Net Send WSAStartup", "Debug" , MB_OK);
                return -1;
        }

        // Creating UDP Socket
        if ((sendsock = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
        {
                MessageBox(NULL,"Failed Net Send socket creation", "Debug" , MB_OK);
                return -1;
        }

        true_int = 1;
        if(setsockopt(sendsock,SOL_SOCKET,SO_REUSEADDR, (char *)&true_int,
sizeof(int))==SOCKET_ERROR)
        {
                closesocket(sendsock);
                MessageBox(NULL,"Failed Net Send set socket option 1", "Debug" , MB_OK);
                return -1;
        }

        usPort = PORT_NUMBER;
        sock_addr.sin_family = AF_INET;
        sock_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        sock_addr.sin_port = htons(usPort);

        send_addr.sin_family = AF_INET;
        send_addr.sin_addr.s_addr = inet_addr(DESTINATION_ADDRESS);
        send_addr.sin_port = htons(usPort);
```

```
	if (bind(sendsock, (LPSOCKADDR) &sock_addr,
		sizeof(sock_addr))==SOCKET_ERROR)
	{
		closesocket(sendsock);
		MessageBox(NULL,"Failed Net Send socket binding", "Debug" , MB_OK);
		return -1;
	}

	if(connect(sendsock, (LPSOCKADDR) &send_addr,
		sizeof(send_addr))==SOCKET_ERROR)
	{

		true_int = 1;
		if(setsockopt(sendsock,SOL_SOCKET,SO_REUSEADDR, (char *)&true_int,
		sizeof(int))==SOCKET_ERROR)
		{
			closesocket(sendsock);
			MessageBox(NULL,"Failed Net Send set socket option 2", "Debug" ,
MB_OK);
			return -1;
		}

		if(connect(sendsock, (LPSOCKADDR) &send_addr,
			sizeof(send_addr))==SOCKET_ERROR)
		{
			closesocket(sendsock);
			error = WSAGetLastError();
			Aud_Strm_Net_Err_Translate(error);
			MessageBox(NULL,"Failed Net Send at connect", "Debug" , MB_OK);
			return -1;
		}
	}

	return 0;
}

void Aud_Strm_Net_Send(unsigned char * buffer)
{
	int tot_sent = 0;
	int ret = 0;

	while(tot_sent!=PACKET_SIZE)
	{
		ret = send(sendsock, (const char *) &buffer[tot_sent], PACKET_SIZE -
tot_sent, 0);
		if(ret==SOCKET_ERROR)
		{
			MessageBox(NULL,"Failed Net Send at send", "Debug" , MB_OK);
			break;
		}
		else tot_sent += ret;
	}
}

void Aud_Strm_Net_Send_Clean(void)
{

	closesocket(sendsock);
	WSACleanup();

	return;
}

void Aud_Strm_Net_Err_Translate(int error)
{

	switch(error)
	{
```

```
case WSAEINTR:
        MessageBox(NULL, "     case WSAEINTR", "NET_ERR", MB_OK);
        break;

case WSAEACCES:
        MessageBox(NULL, "     case WSAEACCES", "NET_ERR", MB_OK);
        break;

case WSAEFAULT:
        MessageBox(NULL, "     case WSAEFAULT", "NET_ERR", MB_OK);
        break;

case WSAEINVAL:
        MessageBox(NULL, "     case WSAEINVAL", "NET_ERR", MB_OK);
        break;

case WSAEMFILE:
        MessageBox(NULL, "     case WSAEMFILE", "NET_ERR", MB_OK);
        break;

case WSAEWOULDBLOCK:
         MessageBox(NULL, "WSAEWOULDBLOCK", "NET_ERR", MB_OK);
        break;

case WSAEINPROGRESS:
         MessageBox(NULL, "WSAEINPROGRESS", "NET_ERR", MB_OK);
        break;

case WSAEALREADY:
         MessageBox(NULL, "WSAEALREADY", "NET_ERR", MB_OK);
        break;

case WSAENOTSOCK:
         MessageBox(NULL, "WSAENOTSOCK", "NET_ERR", MB_OK);
        break;

case WSAEDESTADDRREQ:
         MessageBox(NULL, "WSAEDESTADDRREQ", "NET_ERR", MB_OK);
        break;

case WSAEMSGSIZE:
         MessageBox(NULL, "WSAEMSGSIZE", "NET_ERR", MB_OK);
        break;

case WSAEPROTOTYPE:
         MessageBox(NULL, "WSAEPROTOTYPE", "NET_ERR", MB_OK);
        break;

case WSAENOPROTOOPT:
         MessageBox(NULL, "WSAENOPROTOOPT", "NET_ERR", MB_OK);
        break;

case WSAEPROTONOSUPPORT:
         MessageBox(NULL, "WSAEPROTONOSUPPORT", "NET_ERR", MB_OK);
        break;

case WSAESOCKTNOSUPPORT:
         MessageBox(NULL, "WSAESOCKTNOSUPPORT", "NET_ERR", MB_OK);
        break;

case WSAEOPNOTSUPP:
         MessageBox(NULL, "WSAEOPNOTSUPP", "NET_ERR", MB_OK);
        break;

case WSAEPFNOSUPPORT:
         MessageBox(NULL, "WSAEPFNOSUPPORT", "NET_ERR", MB_OK);
        break;

case WSAEAFNOSUPPORT:
         MessageBox(NULL, "WSAEAFNOSUPPORT", "NET_ERR", MB_OK);
        break;
```

```
case WSAEADDRINUSE:
        MessageBox(NULL, "WSAEADDRINUSE", "NET_ERR", MB_OK);
      break;

case WSAEADDRNOTAVAIL:
        MessageBox(NULL, "WSAEADDRNOTAVAIL", "NET_ERR", MB_OK);
      break;

case WSAENETDOWN:
        MessageBox(NULL, "WSAENETDOWN", "NET_ERR", MB_OK);
      break;

case WSAENETUNREACH:
        MessageBox(NULL, "WSAENETUNREACH", "NET_ERR", MB_OK);
      break;

case WSAENETRESET:
        MessageBox(NULL, "WSAENETRESET", "NET_ERR", MB_OK);
      break;

case WSAECONNABORTED:
        MessageBox(NULL, "WSAECONNABORTED", "NET_ERR", MB_OK);
      break;

case WSAECONNRESET:
        MessageBox(NULL, "WSAECONNRESET", "NET_ERR", MB_OK);
      break;

case WSAENOBUFS:
        MessageBox(NULL, "WSAENOBUFS", "NET_ERR", MB_OK);
      break;

case WSAEISCONN:
        MessageBox(NULL, "WSAEISCONN", "NET_ERR", MB_OK);
      break;

case WSAENOTCONN:
        MessageBox(NULL, "WSAENOTCONN", "NET_ERR", MB_OK);
      break;

case WSAESHUTDOWN:
        MessageBox(NULL, "WSAESHUTDOWN", "NET_ERR", MB_OK);
      break;

case WSAETIMEDOUT:
        MessageBox(NULL, "WSAETIMEDOUT", "NET_ERR", MB_OK);
      break;

case WSAECONNREFUSED:
        MessageBox(NULL, "WSAECONNREFUSED", "NET_ERR", MB_OK);
      break;

case WSAEHOSTDOWN:
        MessageBox(NULL, "WSAEHOSTDOWN", "NET_ERR", MB_OK);
      break;

case WSAEHOSTUNREACH:
        MessageBox(NULL, "WSAEHOSTUNREACH", "NET_ERR", MB_OK);
      break;

case WSAEPROCLIM:
        MessageBox(NULL, "WSAEPROCLIM", "NET_ERR", MB_OK);
      break;

case WSASYSNOTREADY:
        MessageBox(NULL, "WSASYSNOTREADY", "NET_ERR", MB_OK);
      break;

case WSAVERNOTSUPPORTED:
        MessageBox(NULL, "WSAVERNOTSUPPORTED", "NET_ERR", MB_OK);
```

```
            break;

        case WSANOTINITIALISED:
              MessageBox(NULL, "WSANOTINITIALISED", "NET_ERR", MB_OK);
            break;

        case WSAEDISCON:
              MessageBox(NULL, "WSAEDISCON", "NET_ERR", MB_OK);
            break;

/*      case WSATYPE_NOT_FOUND:
              MessageBox(NULL, "WSATYPE_NOT_FOUND", "NET_ERR", MB_OK);
            break;

        case WSAHOST_NOT_FOUND:
              MessageBox(NULL, "WSAHOST_NOT_FOUND", "NET_ERR", MB_OK);
            break;

        case WSATRY_AGAIN:
              MessageBox(NULL, "WSATRY_AGAIN", "NET_ERR", MB_OK);
            break;

        case WSANO_RECOVERY:
              MessageBox(NULL, "WSANO_RECOVERY", "NET_ERR", MB_OK);
            break;

        case WSANO_DATA:
              MessageBox(NULL, "WSANO_DATA", "NET_ERR", MB_OK);
            break;

        case WSA_INVALID_HANDLE:
              MessageBox(NULL, "WSA_INVALID_HANDLE", "NET_ERR", MB_OK);
            break;

        case WSA_INVALID_PARAMETER:
              MessageBox(NULL, "WSA_INVALID_PARAMETER", "NET_ERR", MB_OK);
            break;

        case WSA_IO_INCOMPLETE:
              MessageBox(NULL, "WSA_IO_INCOMPLETE", "NET_ERR", MB_OK);
            break;

        case WSA_IO_PENDING:
              MessageBox(NULL, "WSA_IO_PENDING", "NET_ERR", MB_OK);
            break;

        case WSA_NOT_ENOUGH_MEMORY:
              MessageBox(NULL, "WSA_NOT_ENOUGH_MEMORY", "NET_ERR", MB_OK);
            break;

        case WSA_OPERATION_ABORTED:
              MessageBox(NULL, "WSA_OPERATION_ABORTED", "NET_ERR", MB_OK);
            break;

        case WSAINVALIDPROCTABLE:
            MessageBox(NULL, "WSAINVALIDPROCTABLE", "NET_ERR", MB_OK);
            break;

        case WSAINVALIDPROVIDER:
            MessageBox(NULL, "WSAINVALIDPROVIDER", "NET_ERR", MB_OK);
            break;

        case WSAPROVIDERFAILEDINIT:
            MessageBox(NULL, "WSAPROVIDERFAILEDINIT", "NET_ERR", MB_OK);
            break;

        case WSASYSCALLFAILURE:
            MessageBox(NULL, "WSASYSCALLFAILURE", "NET_ERR", MB_OK);
            break;*/

        default:
                MessageBox(NULL, "???", "NETERR", MB_OK);
                break;

        }

}
```

```
// Aud_Strm_Net_Recv.cpp

#include "stdafx.h"
#include <winsock.h>
#include "Aud_Strm_Net_Recv.h"
#include "Aud_params.h"

static WORD                 wVersionRequested;
static WSADATA          wsaData;
static SOCKET           recvsock;
static SOCKET                   sessionsock;
static char                     chLocalAddress[16];
static struct           sockaddr_in sock_addr;// new_sock_addr
static DWORD            Err;
static u_short          usPort;
static int                      true_int;
static int                      iRet;

int _cdecl DspMsg( HWND hWnd,  HDC hDc,  int x,  int y,  LPSTR msgfmt, ... );

int Aud_Strm_Net_Recv_Init(void)
{
        wVersionRequested = MAKEWORD(2,0);
        iRet = WSAStartup(wVersionRequested, &wsaData);//request winsock version + service
        if (iRet != 0)
        {
                MessageBox(NULL,"Failed Net Recv WSAStartup", "Debug" , MB_OK);
                return -1;
        }

        if ((recvsock = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
        {
                MessageBox(NULL,"Failed Net Recv socket creation", "Debug" , MB_OK);
                return -1;
        }

        true_int = 1;
        if(setsockopt(recvsock,SOL_SOCKET,SO_REUSEADDR, (char *)&true_int,
sizeof(int))==SOCKET_ERROR)
        {
                closesocket(recvsock);
                MessageBox(NULL,"Failed Net Recv set socket option", "Debug" , MB_OK);
                return -1;
        }

        usPort = PORT_NUMBER;
        sock_addr.sin_family = AF_INET;
        sock_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        sock_addr.sin_port = htons(usPort);


        if (bind(recvsock, (LPSOCKADDR) &sock_addr,
                sizeof(sock_addr))==SOCKET_ERROR)
        {
                closesocket(recvsock);
                MessageBox(NULL,"Failed Net Recv socket binding", "Debug" , MB_OK);
                return -1;
        }

        if(listen(recvsock, 1)==SOCKET_ERROR)
        {
                MessageBox(NULL,"Failed Net Recv at listen", "Debug" , MB_OK);
                return -1;
```

```
	}

	sessionsock = accept(recvsock, NULL, NULL);
	if(sessionsock  == INVALID_SOCKET)
	{
		MessageBox(NULL,"Failed Net Recv at accept", "Debug" , MB_OK);
		return -1;
	}


	return 0;
}

void Aud_Strm_Net_Recv_Clean(void)
{
	closesocket(sessionsock);
	closesocket(recvsock);
	WSACleanup();

	return;
}

int Aud_Strm_Net_Recv(unsigned char * buffer)
{
	int tot_recv = 0;
	int ret = 0;

	while(tot_recv!=PACKET_SIZE)
	{
		ret = recv(sessionsock, (char *) &buffer[tot_recv], PACKET_SIZE -
tot_recv, 0);
		if(ret==SOCKET_ERROR)
		{
			MessageBox(NULL,"Failed Net Send at send", "Debug" , MB_OK);
			break;
		}
		else tot_recv += ret;
	}

	return 0;
}
```

// Aud_Stream1.h : main header file for the AUD_STREAM1 application
//

```
#if !defined(AFX_AUD_STREAM1_H__567C9B98_D4FF_42C1_9509_B06A91B91DDC__INCLUDED_)
#define AFX_AUD_STREAM1_H__567C9B98_D4FF_42C1_9509_B06A91B91DDC__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CAud_Stream1App:
// See Aud_Stream1.cpp for the implementation of this class
//

class CAud_Stream1App : public CWinApp
{
public:
	CAud_Stream1App();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAud_Stream1App)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CAud_Stream1App)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_AUD_STREAM1_H__567C9B98_D4FF_42C1_9509_B06A91B91DDC__INCLUDED_)
```

// Aud_Stream1Dlg.h : header file

```
#if !defined(AFX_AUD_STREAM1DLG_H__08731E50_8968_46F6_8F87_BE79DA234B89__INCLUDED_)
#define AFX_AUD_STREAM1DLG_H__08731E50_8968_46F6_8F87_BE79DA234B89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CAud_Stream1Dlg dialog

class CAud_Stream1Dlg : public CDialog
{
// Construction
public:
	CAud_Stream1Dlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	//{{AFX_DATA(CAud_Stream1Dlg)
	enum { IDD = IDD_AUD_STREAM1_DIALOG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAud_Stream1Dlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CAud_Stream1Dlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnRecMic();
	afx_msg void OnPlay();
	afx_msg void OnStop();
	virtual void OnOK();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_AUD_STREAM1DLG_H__08731E50_8968_46F6_8F87_BE79DA234B89__INCLUDED_)
```

//Aud_Strm_DSnd_Cap.h

```
#ifndef AUD_STREAM_CAP
#define AUD_STREAM_CAP

#include "Aud_params.h"

//Release the pointer
#define SAFE_FREE(p) { if(p) { (p)->Release(); (p)=NULL; } }

typedef struct Aud_Enc_Buff
{
        unsigned char sample_bits[PACKET_SIZE];
        unsigned char filled;
}Aud_Enc_Buff_t, * Aud_Enc_Buff_ptr;

typedef struct Aud_DS_Cap_Params
{
        unsigned int  frequency;
        unsigned char bits_sample;
        unsigned char channel;
        Aud_Enc_Buff_ptr fifo_ptr;
        unsigned int done;
        HWND g_hDlg;
}Aud_DS_Cap_Params_t, * Aud_DS_Cap_Params_ptr;

void __cdecl Aud_Strm_DS_Capture(void *);
HRESULT InitDirectSoundCapture( HWND hDlg );
HRESULT CreateCaptureBuffer(void);
HRESULT InitCaptureNotifications(void);
HRESULT FreeDirectSoundCapture(void);

#endif
```

//Aud_Strm_DSnd_Play.h

```
#ifndef AUD_STREAM_PLAY
#define AUD_STREAM_PLAY



void __cdecl Aud_Strm_DS_Play(void * params);
HRESULT InitDirectSoundPlayback( HWND hDlg, int freq, int bits_per_sample, int channels);
HRESULT CreateStreamingPlaybackBuffer(void);
HRESULT InitPlaybackNotifications(void);
HRESULT FreeDirectSoundPlay(void);


#endif
```

//Aud_Strm_Encrypt.h

```
#ifndef AUD_STREAM_ENCRYPT
#define AUD_STREAM_ENCRYPT

#include "Aud_Strm_DSnd_Cap.h"

typedef struct Aud_Enc_Params
{
        Aud_Enc_Buff_ptr global_fifo;
        int max_element_in_fifo;
        int size_of_element;
        unsigned int done;
}Aud_Enc_Params_t, * Aud_Enc_Params_ptr;


#define INIT_CONDITION 0.876786

void Aud_Strm_Bit_Swap_Encrypt_Init(void);
void Aud_Strm_Bit_Swap_Decrypt_Init(void);
void Key_Gen_Encrypt_Decrypt(void);
void Aud_Strm_Bit_Swap_Encrypt_Sample(unsigned char * sample_byte);
void Aud_Strm_Bit_Swap_Decrypt_Sample(unsigned char * sample_byte);
void Aud_Strm_Encrypt_Sample(unsigned char *input_val, unsigned char *output_val);
void Aud_Strm_Decrypt_Sample(unsigned char *input_val, unsigned char *output_val);
void __cdecl Aud_Strm_Encrypt_Fifo_Thread(void * param);
void __cdecl Aud_Strm_Decrypt_Fifo_Thread(void * param);

#endif
```

//Aud_Strm_Net_Recv.h

```
#ifndef AUD_STREAM_NET_RECV
#define AUD_STREAM_NET_RECV

#define NET_RECV_TEST 0

int Aud_Strm_Net_Recv_Init(void);
int Aud_Strm_Net_Recv(unsigned char * buffer);
void Aud_Strm_Net_Recv_Clean(void);

#endif
```

//Aud_Strm_Net_Send.h

```
#ifndef AUD_STREAM_NET_SEND
#define AUD_STREAM_NET_SEND
#define NET_SEND_TEST 0

int Aud_Strm_Net_Send_Init(void);
void Aud_Strm_Net_Send(unsigned char * buffer);
void Aud_Strm_Net_Send_Clean(void);

#endif
```

//Aud_param.h

```
#ifndef AUD_PARAMS
#define AUD_PARAMS
#define PORT_NUMBER 15150
#define DESTINATION_ADDRESS "160.0.108.41"
/* currently system only validated on 11Khz, 8bps, mono*/
#define NUM_PACKETS 25         // 25 notifications per second
#define PACKET_SIZE 441        // 441 per Aud_enc_buff (and network packets)
                               // which equates to about 1/25th of sec
                               // should match num of notifications above
/* with 11Khz Sampling rate at 8 bits per sample in mono, we have 11Kilobytes per second,
11025/25= 441bytes */

#define WAV_FREQ      11025   //sampling frequency
#define WAV_BPS  8       //bits per sample
#define WAV_CHNL 1       //1 = mono, 2 = stereo


#endif
```

APPENDIX E

The M-files attached run the Advanced Encryption Standard (AES) encryption-decryption program on the user's terminal in the MATLAB software. The codes were obtained from http://buchhloz.hs-bremen.de/aes/aes.htm

---

```
%aes_encrypt.m

a=1;
b=16;
fid=fopen('C:\Misc_2\hello.wav', 'r');  %insert filename accordingly
sound_file=fread(fid);

%text1='i must do this properly and i will do it'
%text1=double(text1);
%tlength=length(text1);
%num=ceil(tlength/16);
sound_length=length(sound_file);
num=ceil(sound_length/16);

[s_box, inv_s_box, w, poly_mat, inv_poly_mat] = aes_init;

for i=1:num;
    %y=text1(a:b);
    y=sound_file(a:b);
    %a = a + hex2dec(10);
    %b = b + hex2dec(10);

    n=length(y);
    z=zeros(16,1);
    for m=1:n;
        z(m)=z(m)+y(m);
    end

    plaintext=z;
    ciphertext = cipher (plaintext, w, s_box, poly_mat, 1);
    re_plaintext = inv_cipher (ciphertext, w, inv_s_box, inv_poly_mat, 1);

    if i==1
        mat1=plaintext;
        mat2=re_plaintext;
        mat3=ciphertext;
    else
        mat4=plaintext;
        mat5=re_plaintext;
        mat6=ciphertext;

        original=cat(1,mat1,mat4)
        decrypted=cat(2,mat2,mat5)
        encrypted=cat(2,mat3,mat6)

        mat1=original;
        mat2=decrypted;
        mat3=encrypted;

    end
    a = a + 16;

    b = b + 16;
    if b>sound_length
        b=sound_length;
    end

end
```

```
% aes_init.m

function [s_box, inv_s_box, w, poly_mat, inv_poly_mat] = aes_init
```

```
%AES_INIT  Initialisation of AES-components.
%
%   [S_BOX, INV_S_BOX, W, POLY_MAT, INV_POLY_MAT] = AES_INIT
%   initializes AES-components to be used by subsequent functions.
%   In the initialization step the S-boxes (S_BOX and INV_S_BOX) and the polynomial
%   matrice (POLY_MAT and INV_POLY_MAT) are created and an example cipher key is expanded
%   into the round key schedule (W).

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de
%   Version 1.0     30.05.2001

% Clear the command window
clc

% Create the S-box and the inverse S-box
[s_box, inv_s_box] = s_box_gen (1);

% Create the round constant array
rcon = rcon_gen (1);


%global key
key=randint(1,16,[0 255])

% Create the expanded key (schedule)
w = key_expansion(key, s_box, rcon, 1);


% Create the polynomial transformation matrix and the inverse polynomial matrix
% to be used in MIX_COLUMNS
[poly_mat, inv_poly_mat] = poly_mat_gen (1);
```

```
%aff_trans.m

function b_out = aff_trans (b_in)
%AFF_TRANS  Apply an affine transformation over GF(2^8).
%
%   B_OUT = AFF_TRANS (B_IN)
%   applies an affine transformation to the input byte B_IN.
%
%   The transformation consists of
%   1. a polynomial modulo multiplication
%      by a predefined multiplication polynomial
%      using a predefined modulo polynomial over GF(2^8) and
%   2. the addition (XOR) of a predefined addition polynomial
%
%   B_IN has to be a byte (0 <= B_IN <= 255).

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001


% Define the modulo polynomial
% to be used in the modulo operation in poly_mult
mod_pol = bin2dec ('100000001');

% Define the multiplication polynomial
% In the Rijndael AES Proposal
% they say they use the polynomial '11110001',
% which is wrong.
mult_pol = bin2dec ('00011111');

% Define the addition polynomial
add_pol = bin2dec ('01100011');

% Modular polynomial multiplication
% of the input byte and the fixed multiplication polynomial
temp = poly_mult (b_in, mult_pol, mod_pol);
```

```
% Add (XOR) the constant (addition polynomial)
b_out = bitxor (temp, add_pol);
```

```
%cipher.m

function ciphertext = cipher (plaintext, w, s_box, poly_mat, vargin)
%CIPHER  Convert 16 bytes of plaintext to 16 bytes of ciphertext.
%
%   CIPHERTEXT = CIPHER (PLAINTEXT, W, S_BOX, POLY_MAT)
%   converts PLAINTEXT to CIPHERTEXT,
%   using the expanded cipher key W,
%   the byte substitution table S_BOX, and
%   the transformation matrix POLY_MAT.
%
%   CIPHERTEXT = CIPHER (PLAINTEXT, W, S_BOX, POLY_MAT, 1)
%   switches verbose mode on, which displays intermediate results.
%
%   PLAINTEXT has to be a vector of 16 bytes (0 <= PLAINTEXT(i) <= 255).
%   W has to be a [44 x 4]-matrix of bytes (0 <= W(i,j) <= 255).

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% If there is an optional "verbose mode" argument
if nargin > 4

    % Switch the verbose mode flag on
    verbose_mode = 1;

% If there is no optional "verbose mode" argument
else

    % Switch the verbose mode flag off
    verbose_mode = 0;

end

% If the input vector is a cell array or does not have 16 elements
if iscell (plaintext) | prod (size (plaintext)) ~= 16

    % Inform user and abort
    error ('Plaintext has to be a vector (not a cell array) with 16 elements.')

end

%If any element of the input vector cannot be represented by 8 bits
if any (plaintext < 0 | plaintext > 255)

    % Inform user and abort
    error ('Elements of plaintext vector have to be bytes (0 <= plaintext(i) <= 255).')

end

% If the expanded key array is a cell arrray or does not have the correct size
if iscell (w) | any (size (w) ~= [44, 4])

    % Inform user and abort
    error ('w has to be an array (not a cell array) with [44 x 4] elements.')

end

% If any element of the expanded key array can not be represented by 8 bits
if any (w < 0 | w > 255)

    % Inform user and abort
    error ('Elements of key array w have to be bytes (0 <= w(i,j) <= 255).')

end
```

```
% Display headline if requested
if verbose_mode
    disp (' ')
    disp ('********************************************')
    disp ('*                                          *')
    disp ('*              C I P H E R                 *')
    disp ('*                                          *')
    disp ('********************************************')
    disp (' ')
end

% Copy the 16 elements of the input vector
% column-wise into the 4 x 4 state matrix
state = reshape (plaintext, 4, 4);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Initial state :                    ', state)
end

% Copy the first 4 rows (4 x 4 elements) of the expanded key
% into the current round key.
% Transpose to make this column-wise
round_key = (w(1:4, :))';

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Initial round key :                ', round_key)
end

% Add (xor) the current round key (matrix) to the state (matrix)
state = add_round_key (state, round_key);

% Loop over 9 rounds
for i_round = 1 : 9

    % Display intermediate result if requested
    if verbose_mode
        disp_hex (['State at start of round ', num2str(i_round),' :        '], state)
    end

    % Substitute all 16 elements of the state matrix
    % by shoving them through the S-box
    state = sub_bytes (state, s_box);

    % Display intermediate result if requested
    if verbose_mode
        disp_hex ('After sub_bytes :                  ', state)
    end

    % Cyclically shift the last three rows of the state matrix
    state = shift_rows (state);

    % Display intermediate result if requested
    if verbose_mode
        disp_hex ('After shift_rows :                 ', state)
    end

    % Transform the columns of the state matrix via a four-term polynomial
    state = mix_columns (state, poly_mat);

    % Display intermediate result if requested
    if verbose_mode
        disp_hex ('After mix_columns :                ', state)
    end

    % Extract the current round key (4 x 4 matrix) from the expanded key
    round_key = (w((1:4) + 4*i_round, :))';

    % Display intermediate result if requested
    if verbose_mode
```

```
        disp_hex ('Round key :                      ', round_key)
    end

    % Add (XOR) the current round key (matrix) to the state (matrix)
    state = add_round_key (state, round_key);

end

% Display intermediate result if requested
if verbose_mode
    disp_hex ('State at start of final round :  ', state)
end

% Substitute all 16 elements of the state matrix
% by shoving them through the S-box
state = sub_bytes (state, s_box);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('After sub_bytes :                ', state)
end

% Cyclically shift the last three rows of the state matrix
state = shift_rows (state);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('After shift_rows :               ', state)
end

% Extract the last round key (4 x 4 matrix) from the expanded key
round_key = (w(41:44, :))';

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Round key :                      ', round_key)
end

% Add (xor) the current round key (matrix) to the state (matrix)
state = add_round_key (state, round_key);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Final state :                    ', state)
end

% reshape the 4 x 4 state matrix into a 16 element row vector
ciphertext = reshape (state, 1, 16);
```

```
%inv_cipher.m

function plaintext = inv_cipher (ciphertext, w1, inv_s_box, inv_poly_mat, vargin)
%INV_CIPHER  Convert 16 bytes of ciphertext to 16 bytes of plaintext.
%
%   PLAINTEXT = INV_CIPHER (CIPHERTEXT, W, INV_S_BOX, INV_POLY_MAT)
%   converts CIPHERTEXT (back) to the plaintext PLAINTEXT,
%   using the expanded cipher key W,
%   the inverse byte substitution table INV_S_BOX, and
%   the inverse transformation matrix INV_POLY_MAT.
%
%   PLAINTEXT = INV_CIPHER (CIPHERTEXT, W, INV_S_BOX, INV_POLY_MAT, 1)
%   switches verbose mode on, that displays intermediate results.

%   CIPHERTEXT has to be a vector of 16 bytes (0 <= CIPHERTEXT(i) <= 255).
%   W has to be a [44 x 4]-matrix of bytes (0 <= W(i,j) <= 255).

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001
% If there is an optional "verbose mode" argument
if nargin > 4

    % Switch the verbose mode flag on
    verbose_mode = 1;

% If there is no optional "verbose mode" argument
else

    % Switch the verbose mode flag off
    verbose_mode = 0;

end

% If the input vector is a cell array or does not have 16 elements
if iscell (ciphertext) | prod (size (ciphertext)) ~= 16

    % Inform user and abort
    error ('Ciphertext has to be a vector (not a cell array) with 16 elements.')

end

% If any element of the input vector cannot be represented by 8 bits
if any (ciphertext < 0 | ciphertext > 255)

    % Inform user and abort
    error ('Elements of ciphertext vector have to be bytes (0 <= ciphertext(i) <= 255).')

end

% If the expanded key array is a cell arrray or does not have the correct size
if iscell (w1) | any (size (w1) ~= [44, 4])

    % Inform user and abort
    error ('w has to be an array (not a cell array) with [44 x 4] elements.')

end

% If any element of the expanded key array can not be represented by 8 bits
if any (w1 < 0 | w1 > 255)

    % Inform user and abort
    error ('Elements of key array w1 have to be bytes (0 <= w1(i,j) <= 255).')

end

% Display headline if requested
if verbose_mode
    disp (' ')
    disp ('********************************************')
    disp ('*                                          *')
    disp ('*      I N V E R S E   C I P H E R         *')
    disp ('*                                          *')
    disp ('********************************************')
    disp (' ')
end

% Copy the 16 elements of the input vector column-wlise into the 4 x 4 state matrix
state = reshape (ciphertext, 4, 4);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Initial state :                    ', state)
end

% Copy the last 4 rowls (4 x 4 elements) of the expanded key
% into the current round key.
% Transpose to make this column-wlise
round_key = (w1(41:44, :))';

% Display intermediate result if requested
```

```
if verbose_mode
    disp_hex ('Initial round key :              ', round_key)
end

% Add (xor) the current round key (matrix) to the state (matrix)
state = add_round_key (state, round_key);

% Loop over 9 rounds backwlards
for i_round = 9 : -1 : 1

    % Display intermediate result if requested
    if verbose_mode
        disp_hex (['State at start of round ', num2str(i_round),' :      '], state)
    end

    % Cyclically shift the last three rowls of the state matrix
    state = inv_shift_rows (state);

    % Display intermediate result if requested
    if verbose_mode
        disp_hex ('After inv_shift_rowls :           ', state)
    end

    % Substitute all 16 elements of the state matrix
    % by shoving them through the S-box
    state = sub_bytes (state, inv_s_box);

    % Display intermediate result if requested
    if verbose_mode
        disp_hex ('After inv_sub_bytes :             ', state)
    end

    % Extract the current round key (4 x 4 matrix) from the expanded key
    round_key = (w1((1:4) + 4*i_round, :))';

    % Display intermediate result if requested
    if verbose_mode
        disp_hex ('Round key :                       ', round_key)
    end

    % Add (XOR) the current round key (matrix) to the state (matrix)
    state = add_round_key (state, round_key);

    % Display intermediate result if requested
    if verbose_mode
        disp_hex ('After add_round_key :             ', state)
    end

    % Transform the columns of the state matrix via a four-term polynomial.
    % Use the same function (mix_columns) as in cipher,
    % but wlith the inverse polynomial matrix
    state = mix_columns (state, inv_poly_mat);

end

% Display intermediate result if requested
if verbose_mode
    disp_hex ('State at start of final round :  ', state)
end

% Cyclically shift the last three rowls of the state matrix
state = inv_shift_rows (state);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('After inv_shift_rows :           ', state)
end

% Substitute all 16 elements of the state matrix
% by shoving them through the inverse S-box
state = sub_bytes (state, inv_s_box);
```

```
% Display intermediate result if requested
if verbose_mode
    disp_hex ('After inv_sub_bytes :              ', state)
end

% Extract the "first" (final) round key (4 x 4 matrix) from the expanded key
round_key = (w1(1:4, :))';

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Round key :                        ', round_key)
end

% Add (xor) the current round key (matrix) to the state (matrix)
state = add_round_key (state, round_key);

% Display intermediate result if requested
if verbose_mode
    disp_hex ('Final state :                      ', state)
end

% reshape the 4 x 4 state matrix into a 16 element row vector
plaintext = reshape (state, 1, 16);
```

```
%cycle.m

function matrix_out = cycle (matrix_in, direction)
%SHIFT_ROWS  Cyclically shift the rows of the state matrix.
%
%   MATRIX_OUT = CYCLE (MATRIX_IN, 'left')
%   cyclically shifts the last three rows of the input matrix to the left.
%   The first row is not shifted:                        [1 2 3 4]
%   The second row is cyclically shifted once to the left:   [2 3 4 1]
%   The third row is cyclically shifted twice to the left:   [3 4 1 2]
%   The fourth row is cyclically shifted thrice to the left: [4 1 2 3]

%   MATRIX_OUT = CYCLE (MATRIX_IN, 'right')
%   cyclically shifts the last three rows of the input matrix to the right.
%   The first row is not shifted:                         [1 2 3 4]
%   The second row is cyclically shifted once to the right:   [4 1 2 3]
%   The third row is cyclically shifted twice to the right:   [3 4 1 2]
%   The fourth row is cyclically shifted thrice to the right: [2 3 4 1]

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% If the matrix has to be shifted to the left,
if strcmp (direction, 'left')

    % generate the column vector [0 5 10 15]'
    col = (0 : 5 : 15)';

% If the matrix has to be shifted to the right,
else

    % generate the column vector [16 13 10 7]'
    col = (16 : -3 : 7)';

end

% Generate the row vector [0 4 8 12]
row = 0 : 4 : 12;

% Repeat the column to create the matrix [ 0  0  0  0] (left shift)
%                                        [ 5  5  5  5]
%                                        [10 10 10 10]
%                                        [15 15 15 15]
cols = repmat (col, 1, 4);
```

```
% Repeat the row to create the matrix [0 4 8 12]
%                                     [0 4 8 12]
%                                     [0 4 8 12]
%                                     [0 4 8 12]
rows = repmat (row, 4, 1);

% Add both matrices,
% fold back into the 0 ... 15 domain,
% and add 1, because Matlab indices do start with 1
% [ 1  5  9 13]
% [ 6 10 14  2]
% [11 15  3  7]
% [16  4  8 12]
ind_mat = mod (rows + cols, 16) + 1;

% Apply the just created index matrix to the input matrix.
% Elements of the index matrix are linear (column-wise) indices.
matrix_out = matrix_in (ind_mat);
```

```
%find_inverse.m

function b_inv = find_inverse (b_in, mod_pol)
%FIND_INVERSE  Find the multiplicative inverse in GF(2^8).
%
%   B_INV = FIND_INVERSE (B_IN, MOD_POL)
%   finds the multiplicative inverse of B_IN
%   in the finite Galois field GF(2^8)
%   with respect to the predefined (irreducible modulo polynomial.
%
%   B_IN has to be a byte (0 <= B_IN <= 255).
%
%   This implementation is extremely simple, uneconomic, and slow;
%   but it works and clearly demonstrates the definition of the inverse.
%   Smarter implementations e.g. use the "extended Euclidean algorithm".

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Loop over all possible "test" bytes.
% The inverse of zero is defined as zero
for i = 1 : 255

    % "Test-wise" compute the polynomial multiplication
    % of the input byte and the current "test byte"
    prod = poly_mult (b_in, i, mod_pol);

    % If the polynomial modulo multiplication leaves a remainder of "1"
    % we have found the inverse
    if prod == 1

        % Declare (save and return) the current test byte as inverse,
        b_inv = i;

        % and abort the search
        break
    end
end
```

```
%key_expansion.m

function w = key_expansion (key, s_box, rcon, vargin)
%KEY_EXPANSION  Expand the 16-byte cipher key.
%
%   W = KEY_EXPANSION (KEY, S_BOX, RCON)
%   creates the 44x4-byte expanded key W,
```

```
% Repeat the row to create the matrix [0 4 8 12]
%                                     [0 4 8 12]
%                                     [0 4 8 12]
%                                     [0 4 8 12]
rows = repmat (row, 4, 1);

% Add both matrices,
% fold back into the 0 ... 15 domain,
% and add 1, because Matlab indices do start with 1
% [ 1  5  9 13]
% [ 6 10 14  2]
% [11 15  3  7]
% [16  4  8 12]
ind_mat = mod (rows + cols, 16) + 1;

% Apply the just created index matrix to the input matrix.
% Elements of the index matrix are linear (column-wise) indices.
matrix_out = matrix_in (ind_mat);
```

```
%find_inverse.m

function b_inv = find_inverse (b_in, mod_pol)
%FIND_INVERSE  Find the multiplicative inverse in GF(2^8).
%
%   B_INV = FIND_INVERSE (B_IN, MOD_POL)
%   finds the multiplicative inverse of B_IN
%   in the finite Galois field GF(2^8)
%   with respect to the predefined (irreducible modulo polynomial.
%
%   B_IN has to be a byte (0 <= B_IN <= 255).
%
%   This implementation is extremely simple, uneconomic, and slow;
%   but it works and clearly demonstrates the definition of the inverse.
%   Smarter implementations e.g. use the "extended Euclidean algorithm".

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Loop over all possible "test" bytes.
% The inverse of zero is defined as zero
for i = 1 : 255

    % "Test-wise" compute the polynomial multiplication
    % of the input byte and the current "test byte"
    prod = poly_mult (b_in, i, mod_pol);

    % If the polynomial modulo multiplication leaves a remainder of "1"
    % we have found the inverse
    if prod == 1

        % Declare (save and return) the current test byte as inverse,
        b_inv = i;

        % and abort the search
        break
    end
end
```

```
%key_expansion.m

function w = key_expansion (key, s_box, rcon, vargin)
%KEY_EXPANSION  Expand the 16-byte cipher key.
%
%   W = KEY_EXPANSION (KEY, S_BOX, RCON)
%   creates the 44x4-byte expanded key W,
% Repeat the row to create the matrix [0 4 8 12]
%                                     [0 4 8 12]
%                                     [0 4 8 12]
%                                     [0 4 8 12]
rows = repmat (row, 4, 1);

% Add both matrices,
% fold back into the 0 ... 15 domain,
% and add 1, because Matlab indices do start with 1
% [ 1  5  9 13]
% [ 6 10 14  2]
% [11 15  3  7]
% [16  4  8 12]
ind_mat = mod (rows + cols, 16) + 1;

% Apply the just created index matrix to the input matrix.
% Elements of the index matrix are linear (column-wise) indices.
matrix_out = matrix_in (ind_mat);
```

```
%find_inverse.m

function b_inv = find_inverse (b_in, mod_pol)
%FIND_INVERSE  Find the multiplicative inverse in GF(2^8).
%
%   B_INV = FIND_INVERSE (B_IN, MOD_POL)
%   finds the multiplicative inverse of B_IN
%   in the finite Galois field GF(2^8)
%   with respect to the predefined (irreducible modulo polynomial.
%
%   B_IN has to be a byte (0 <= B_IN <= 255).
%
%   This implementation is extremely simple, uneconomic, and slow;
%   but it works and clearly demonstrates the definition of the inverse.
%   Smarter implementations e.g. use the "extended Euclidean algorithm".

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Loop over all possible "test" bytes.
% The inverse of zero is defined as zero
for i = 1 : 255

    % "Test-wise" compute the polynomial multiplication
    % of the input byte and the current "test byte"
    prod = poly_mult (b_in, i, mod_pol);

    % If the polynomial modulo multiplication leaves a remainder of "1"
    % we have found the inverse
    if prod == 1

        % Declare (save and return) the current test byte as inverse,
        b_inv = i;

        % and abort the search
        break
    end
end
```

```
%key_expansion.m

function w = key_expansion (key, s_box, rcon, vargin)
%KEY_EXPANSION  Expand the 16-byte cipher key.
%
%   W = KEY_EXPANSION (KEY, S_BOX, RCON)
%   creates the 44x4-byte expanded key W,
%   using the initial 16-byte cipher KEY,
%   the predefined byte substitution table S_BOX, and
%   the round constant RCON to be added to every fourth 16-byte sub-key.
%
%   W = KEY_EXPANSION (KEY, S_BOX, RCON, 1)
%   switches verbose mode on, that displays intermediate results.
%
%   KEY has to be a vector of 16 bytes (0 <= KEY(i) <= 255).
%
%   KEY_EXPANSION has to be called prior to CIPHER and INV_CIPHER.

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% If there is an optional "verbose mode" argument
if nargin > 3

    % Switch the verbose mode flag on
    verbose_mode = 1;

% If there is no optional "verbose mode" argument
else

    % Switch the verbose mode flag off
    verbose_mode = 0;

end

% If the key vector is a cell array or does not have 16 elements
if iscell (key) | prod (size (key)) ~= 16

    % Inform user and abort
    error ('Key has to be a vector (not a cell array) with 16 elements.')

end

% If any element of the key vector cannot be represented by 8 bits
if any (key < 0 | key > 255)

    % Inform user and abort
    error ('Elements of key vector have to be bytes (0 <= key(i) <= 255).')

end

% Display headline if requested
if verbose_mode
    disp (' ')
    disp ('********************************************')
    disp ('*                                          *')
    disp ('*        K E Y   E X P A N S I O N         *')
    disp ('*                                          *')
    disp ('********************************************')
    disp (' ')
end

% Copy the 16 elements of the key vector row-wise
% into the first four rows of the expanded key
w = (reshape (key, 4, 4))';

% Display intermediate result if requested
if verbose_mode
    disp_hex ('w(1:4, :) :        ', w)
end

% Loop over the rest of the 44 rows of the expanded key
for i = 5 : 44

    % Copy the previous row of the expanded key into a buffer
    temp = w(i - 1, :);
    % Every fourth row is treated differently:
    if mod (i, 4) == 1

        % Perform a cyclic (byte-wise) permutation to the buffer
        temp = rot_word (temp);

        % Display intermediate result if requested
        if verbose_mode
            disp_hex (['After rot_word :  '], temp)
        end

        % Substitute all 4 elements of the buffer
        % by shoving them through the S-box
        temp = sub_bytes (temp, s_box);

        % Display intermediate result if requested
        if verbose_mode
            disp_hex (['After sub_bytes : '], temp)
        end

        % Compute the current round constant
        r = rcon ((i - 1)/4, :);

        % Display intermediate result if requested
        if verbose_mode
            disp_hex (['rcon(', num2str(i,'%02d'), ', :) :      '], r)
        end

        % Add (XOR) the current rount constant
        % to every element of the buffer
        temp = bitxor (temp, r);

        % Display intermediate result if requested
        if verbose_mode
            disp_hex (['After rcon xor :  '], temp)
        end

    end

    % The new row of the expanded key
    % is the sum (XOR) of the row four rows before
    % and the buffer
    w(i, :) = bitxor (w(i - 4, :), temp);

    % Display intermediate result if requested
    if verbose_mode
        disp_hex (['w(', num2str(i,'%02d'), ', :) :        '], w(i, :))
    end

end
```

```
%add_round_key.m

function state_out = add_round_key (state_in, round_key)
%ADD_ROUND_KEY  Add (XOR) the round key to the state.
%
%   STATE_OUT = ADD_ROUND_KEY (STATE_IN, ROUND_KEY)
%   adds the current round key matrix ROUND_KEY
%   to the current state matrix STATE_IN.
%   Adding in GF(2^8) is performed via bitwise XOR.

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Add state (matrix) and round key (matrix) via bitwise XOR
state_out = bitxor (state_in, round_key);
```

```
%mix_columns.m

function state_out = mix_columns (state_in, poly_mat)
%MIX_COLUMNS  Transform each column of the state matrix.
%
%   STATE_OUT = MIX_COLUMNS (STATE_IN, POLY_MAT)
%   operates on the state matrix STATE_IN column-by-column
%   using POLY_MAT as the transformation matrix.
%
%   MIX_COLUMNS can also directly compute
%   the inverse column transformation INV_MIX_COLUMNS
%   by utilizising the inverse transformation matrix INV_POLY_MAT.

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Define the irreducible polynomial
% to be used in the modulo operation in poly_mult
mod_pol = bin2dec ('100011011');

% Loop over all columns of the state matrix
for i_col_state = 1 : 4

    % Loop over all rows of the state matrix
    for i_row_state = 1 : 4

        % Initialize the scalar product accumulator
        temp_state = 0;

        % For the (innner) matrix vector product we want to do
        % a scalar product
        % of the current row vector of poly_mat
        % and the current column vector of the state matrix.
        % Therefore we need a counter over
        % all elements of the current row vector of poly_mat and
        % all elements of the current column vector of the state matrix
        for i_inner = 1 : 4

            % Multiply (GF(2^8) polynomial multiplication)
            % the current element of the current row vector of poly_mat with
            % the current element of the current column vector of the state matrix
            temp_prod = poly_mult (...
                        poly_mat(i_row_state, i_inner), ...
                        state_in(i_inner, i_col_state), ...
                        mod_pol);

            % Add (XOR) the recently calculated product
            % to the scalar product accumulator
            temp_state = bitxor (temp_state, temp_prod);

        end

        % Declare (save and return) the final scalar product accumulator
        % as the current state matrix element
        state_out(i_row_state, i_col_state) = temp_state;

    end
end
```

```
%shift_rows.m

function state_out = shift_rows (state_in)
%SHIFT_ROWS  Cyclically shift the rows of the state matrix.
%
%   STATE_OUT = SHIFT_ROWS (STATE_IN)
%   cyclically shifts the last three rows of the state matrix to the left.
%   The first row is not shifted:                         [1 2 3 4]
%   The second row is cyclically shifted once to the left:  [2 3 4 1]
%   The third row is cyclically shifted twice to the left:  [3 4 1 2]
%   The fourth row is cyclically shifted thrice to the left: [4 1 2 3]

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Call the function cycle to do the actual left shifting
state_out = cycle (state_in, 'left');
```

```
%inv_shit_rows.m
function state_out=inv_shift_rows (state_in)

%INV_SHIFT_ROWS  Cyclically shift (back) the rows of the state matrix.
%
%   STATE_OUT = INV_SHIFT_ROWS (STATE_IN)
%   cyclically shifts the last three rows fo the state matrix to the right.
%   The first row is not shifted:                           [1 2 3 4]
%   The second row is cyclically shifted once to the right:  [4 1 2 3]
%   The third row is cyclically shifted twice to the right:  [3 4 1 2]
%   The fourth row is cyclically shifted thrice to the right: [2 3 4 1]

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Call the function cycle to do the actual right shifting
state_out = cycle (state_in, 'right');
```

```
%poly_mat_gen.m

function [poly_mat, inv_poly_mat] = poly_mat_gen (vargin)
%POLY_MAT  Create polynomial coefficient matrices.
%
%   [POLY_MAT, INV_POLY_MAT] = POLY_MAT_GEN
%   creates the polynomial coefficient matrices
%   to be used by the function MIX_COLUMNS.
%
%   [POLY_MAT, INV_POLY_MAT] = POLY_MAT_GEN (1)
%   switches verbose mode on, that displays intermediate results.
%
%   POLY_MAT_GEN has to be called prior to CIPHER and INV_CIPHER.

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% If there is an optional "verbose mode" argument
if nargin > 0

    % Switch the verbose mode flag on
    verbose_mode = 1;

% If there is no optional "verbose mode" argument
else

    % Switch the verbose mode flag off
    verbose_mode = 0;

end

% Display headline if requested
if verbose_mode
    disp (' ')
    disp ('*********************************************')
    disp ('*                                           *')
    disp ('*    P O L Y _ M A T   C R E A T I O N      *')
    disp ('*                                           *')
    disp ('*********************************************')
    disp (' ')
end

% Define the first row of the polynomial coefficient matrix
% to be used in MIX_COLUMNS in hexadecimal representation.
% Small values are chosen for computational speed reasons
row_hex = {'02' '03' '01' '01'};

% Convert the polynomial coefficients to decimal "numbers"
% row = [2 3 1 1]
row = hex2dec (row_hex)';

% Construct a matrix with identical rows
% rows = [2 3 1 1]
%        [2 3 1 1]
%        [2 3 1 1]
%        [2 3 1 1]
rows = repmat (row, 4, 1);

% Construct the polynomial matrix
% by cyclically permuting the rows to the right
% poly_mat = [2 3 1 1]
%            [1 2 3 1]
%            [1 1 2 3]
%            [3 1 1 2]
poly_mat = cycle (rows, 'right');

% Define the first row of the inverse polynomial coefficient matrix
% to be used in INV_MIX_COLUMNS in hexadecimal representation.
inv_row_hex = {'0e' '0b' '0d' '09'};

% Convert the polynomial coefficients to decimal "numbers"
inv_row = hex2dec (inv_row_hex)';

% Construct a matrix with identical rows
inv_rows = repmat (inv_row, 4, 1);

% Construct the polynomial matrix
inv_poly_mat = cycle (inv_rows, 'right');

% Display intermediate result if requested
if verbose_mode
    disp_hex ('    poly_mat : ', poly_mat)
    disp_hex ('inv_poly_mat : ', inv_poly_mat)
end
```

```
%poly_mult.m
function ab = poly_mult (a, b, mod_pol)
%POLY_MULT  Polynomial modulo multiplication in GF(2^8).
%
%   AB = POLY_MULT (A, B, MOD_POL)
%   performs a polynomial multiplication of A and B
%   in the finite Galois field GF(2^8),
%   using MOD_POL as the irreducible modulo polynomial.
%
%   A and B have to be bytes (0 <= A, B <= 255).
%   MOD_POL is of degree 8.

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Initialize the product term
% to be used on the right-hand side of the XOR-iteration
ab = 0;

% Loop over every bit of the first factor ("a")
% starting with the least significant bit.
% This loop multiplies "a" and "b" modulo 2
```

```
for i_bit = 1 : 8

    % If the current bit is set,
    % the second factor ("b") has to be multiplied
    % by the corresponding power of 2
    if bitget (a, i_bit)

        % The power-2-multiplication is carried out
        % by the corresponding left shift of the second factor ("b"),
        b_shift = bitshift (b, i_bit - 1);

        % and the modulo 2 (XOR) "addition" of the shifted factor
        ab = bitxor (ab, b_shift);

    end

end

% Loop over the 8 most significant bits of the "ab"-product.
% This loop reduces the 16-bit-product back to the 8 bits
% of a GF(2^8) element by the use of
% the irreducible modulo polynomial of degree 8.
for i_bit = 16 : -1 : 9

    % If the current bit is set,
    % "ab" (or the reduced "ab" respectively) has to be "divided"
    % by the modulo polynomial
    if bitget (ab, i_bit)

        % The "division" is carried out
        % by the corresponding left shift of the modulo polynomial,
        mod_pol_shift = bitshift (mod_pol, i_bit - 9);

        % and the "subtraction" of the shifted modulo polynomial.
        % Since both "addition" and "subtraction" are
        % operations modulo 2 in this context,
        % both can be achieved via XOR
        ab = bitxor (ab, mod_pol_shift);

    end
end
```

```
%rcon_gen.m

function rcon = rcon_gen (vargin)
%RCON_GEN  Create round constants.
%
%   RCON = RCON_GEN
%   creates the round constants vector RCON
%   to be used by the function KEY_EXPANSION.
%
%   RCON = RCON_GEN (1)
%   switches verbose mode on, that displays intermediate results.
%
%   RCON_GEN has to be called prior to KEY_EXPANSION.

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% If there is an optional "verbose mode" argument
if nargin > 0

    % Switch the verbose mode flag on
    verbose_mode = 1;

% If there is no optional "verbose mode" argument
else

    % Switch the verbose mode flag off
    verbose_mode = 0;

end

% Display headline if requested
if verbose_mode
    disp (' ')
    disp ('*******************************************')
    disp ('*                                         *')
    disp ('*        R C O N   C R E A T I O N        *')
    disp ('*                                         *')
    disp ('*******************************************')
    disp (' ')
end

% Define the irreducible polynomial
% to be used in the modulo operation in poly_mult
mod_pol = bin2dec ('100011011');

% The (first byte of the) first round constant is a "1"
rcon(1) = 1;

% Loop over the rest of the elements of the round constant vector
for i = 2 : 10

    % The next round constant is twice the previous one; modulo
    rcon(i) = poly_mult (rcon(i-1), 2, mod_pol);

end

% The other (LSB) three bytes of all round constants are zeros
rcon = [rcon(:), zeros(10, 3)];

% Display intermediate result if requested
if verbose_mode
    disp_hex ('rcon : ', rcon)
end
```

```
%rot_word.m

function w_out = rot_word (w_in)
%ROT_WORD  Rotate the elements of a four element vector.
%
%   W_OUT = ROT_WORD (W_IN)
%   performs a cyclic shift of the elements
%   of the four element vector W_IN.
%   [a1, a2, a3, a4] --> [a2, a3, a4, a1]

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Do the shift...
w_out = w_in([2 3 4 1]);
```

```
%s_box_gen.m

function [s_box, inv_s_box] = s_box_gen (vargin)
%S_BOX_GEN  Create S-box and inverse S-box.
%
%   [S_BOX, INV_S_BOX] = S_BOX_GEN
%   creates the S-box and the inverse S-box
%   to be used by the function SUB_BYTES.
%   The S-box is created in two steps:
%   1. Take the multiplicative inverse of the finite field GF(2^8).
%   2. Apply an affine transformation.
%
%   [S_BOX, INV_S_BOX] = S_BOX_GEN (1)
%   switches verbose mode on, that displays intermediate results.
%
%   S_BOX_GEN has to be called prior to
%   KEY_EXPANSION, CIPHER, and INV_CIPHER.
%
%   In the AES Specification Standard the S-boxes are depicted
%   as arrays. For the sake of indexing-simplicity they are internally
%   stored as vectors in this implementation.

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% If there is an optional "verbose mode" argument
if nargin > 0

    % Switch the verbose mode flag on
    verbose_mode = 1;

% If there is no optional "verbose mode" argument
else

    % Switch the verbose mode flag off
    verbose_mode = 0;

end

% Display headline if requested
if verbose_mode
    disp (' ')
    disp ('********************************************')
    disp ('*                                          *')
    disp ('*      S - B O X   C R E A T I O N         *')
    disp ('*                                          *')
    disp ('*   (this might take a few seconds ;-))    *')
    disp ('*                                          *')
    disp ('********************************************')
    disp (' ')
end

% Define the irreducible polynomial
% to be used in the modulo operation in poly_mult,
% called by find_inverse
mod_pol = bin2dec ('100011011');

% The polynomial multiplicative inverse of zero is defined here as zero.
% Matlab vectors start with an index of "1"
inverse(1) = 0;

% Loop over all remaining byte values
for i = 1 : 255

    % Compute the multiplicative inverse of the current byte value
    % with respect to the specified modulo polynomial
    inverse(i + 1) = find_inverse (i, mod_pol);

end

% Loop over all byte values
for i = 1 : 256

    % Apply the affine transformation
    s_box(i) = aff_trans (inverse(i));

end

% Create the inverse S-box by taking the values
% of the elements of the S-Box as indices:
inv_s_box = s_box_inversion (s_box);

% Display intermediate result if requested
```

```
if verbose_mode

    % Display the s_box and the inverse s_box in 16x16 matrix format.
    % Notice the transpose character for row-wise matrix representation
    s_box_mat = reshape (s_box, 16, 16)';
    disp_hex ('    s_box : ', s_box_mat)
    inv_s_box_mat = reshape (inv_s_box, 16, 16)';
    disp_hex ('inv_s_box : ', inv_s_box_mat)

end
```

```
%s_box_inversion.m
function inv_s_box = s_box_inversion (s_box)
%S_BOX_INVERSION  Invert S-box.
%
%   [INV_S_BOX] = S_BOX_INVERSION (S_BOX)
%   creates the inverse S-box
%   from the previously created S-box.

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001


% Loop over all byte values
for i = 1 : 256

    % Create the inverse S-box by taking the values
    % of the elements of the S-Box as indices:
    % e.g.: s_box(00hex) = 63hex   ==>   inv_s_box(63hex) = 00hex
    % (except the fact, that Matlab vectors start at 1...)
    inv_s_box(s_box(i) + 1) = i - 1;

end
```

```
%sub_bytes.m

function bytes_out = sub_bytes (bytes_in, s_box)
%SUB_BYTES  Nonlinear byte substitution using a substitution table.
%
%   BYTES_OUT = SUB_BYTES (BYTES_IN, S_BOX)
%   transforms the input array BYTES_IN
%   into the output array BYTES_OUT
%   using the substitution table S_BOX.
%
%   BYTES_IN has to be an array of bytes (0 <= BYTES_IN(i) <= 255).

%   Copyright 2001-2005, J. J. Buchholz, Hochschule Bremen, buchholz@hs-bremen.de

%   Version 1.0     30.05.2001

% Thanks to Matlab's marvellous matrix manipulation mastery,
% the substitution of a whole array can be formulated
% in just one statement
bytes_out = s_box (bytes_in + 1);
```