Real Time Implementation Of The Advanced Encryption Standard On DSK 6711

BY

GC HASAN ANSARI (Syndicate LEADER)
GC IKRAM HUSSAIN
PC HAMMAD HAFEEZ

PROJECT DS : Maj (R) Fazal Ahmed

Dissertation Submitted for partial fulfilment of the requirements of MCS/NUST for the award of B.E degree in Telecommunication Engineering.

Department of Electrical Engineering
Military College of Signals
Rawalpindi

APRIL, 2004

TABLE OF CONTENTS

CHAPTER 3 EVOLUTION OF THE ADVANCED ENCRYPTION STANDARDS (AES)

CHAPTER 4 The Advanced Encryption Standard

CHAPTER 5 Design Of the project

List of Figures

CHAPTER 1

Project Overview.

1.1 Introduction

The growth of the communications market and the associated need for information integrity and secrecy encourages the development of secure communications systems. Communication security is especially of greater interest for military purposes. Whatever measures be adopted to provide communication security, they always face the looming danger of being breached by hostile sources. Hence the need to implement a fool proof security measure remains a difficult task. The Project undertaken is an endeavor in the same direction. The aim of the project is to develop a secure communication system over wireless and wire line media with optimum guaranteed security. For this purpose it was required to implement an established encryption algorithm. Developing of encryption algorithm is a gigantic task. The field of cryptanalysis aided by the modern high power computers has further complicated the task of cryptologists. The recent history has shown that many encryption algorithms made by very highly rated firms were cracked by very low budget organization in a matter of days. Most of the speech encryption algorithms remain unpublished hence there were two options available either to develop an encryption algorithm or to tailor one of the published encryption algorithms to our own needs. We chose the second option. In the initial phase of the project various published algorithms were studied. Our task was simplified due to a competition held by NIST for selection of advanced encryption Standard. The winner of this competition was

"Rijndael" an encryption algorithm designed by two Belgians. The evolution of AES is described in chapter 2 of this document. The second important issue was the selection of the platform on which we had to implement our project. The DSK 6711 available in the DSP lab were an instant choice due to there suitability for DSP applications. However AES was a computational intensive algorithm therefore we used a twofold approach and also implemented the secure communication system using PCs with windows platform. The design phase was completed using MATLAB which is described in Chapter 3 of this document. Finally the algorithm was coded in C and implemented on DSK 6711 and on the PC.

1.2 Overview.

The goal of the Project was to develop a secure communication system to be inserted between the information source and the non-secure channel thereby providing a secure channel.

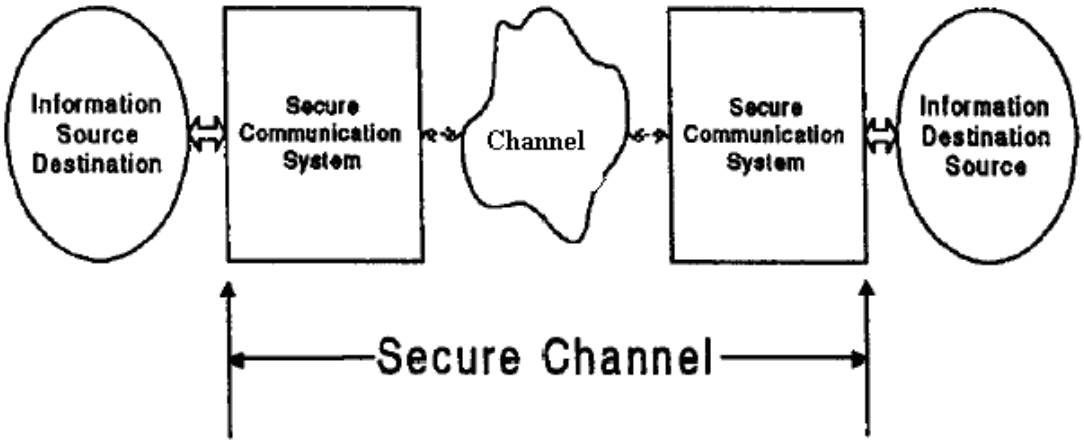


Figure 1.1 Project Overview.

Figure on the previous page shows the modular diagram of the project that we undertook. The information from the information source had to be input to a secure communication system. The main project effort was concentrated in designing and

implementing this module. The emphasis was kept on making the communication security unquestionable. The parameter which determines the security of the channel is the cryptographic algorithm being put into use. The Advanced encryption standard that has been implemented is arguably the most secure in the world. The encrypted information was put onto the channel and received at the other end. The information was decrypted and then output to the information destination. The channel which is otherwise insecure was transformed to a secure channel with a fool proof cryptographic technique. The main challenge in the project was its real time implementation.

1.3 The Secure Communication Model.

Various security algorithms have different structures. This section explains the basic structure of the cryptographic algorithm that has been implemented so as to give the reader the idea as to how encryption and decryption has been achieved.

Figure 2 shows the secure communication model. At the source side there are two parameters that are input to the encrypter i.e.

- Message and
- Encryption key

The message M is encrypted (ciphered) by a transformation governed by the algorithm specified by the Advanced Encryption Standard. The encryption key is used to encrypt the message. The message is then sent to its destination where it is decrypted using the same key as used for the encryption. The message M is obtained by application of the inverse transformation, at the destination. It is essential that the key should be same, even a one bit difference in the key at the encrypter and decrypter would yield undesirable results.

The cryptanalyst algorithm tries to estimate M from the knowledge of the encrypted message. However due to the ignorance of the encryption key fails to decrypt the message. The security of the system resides only in the secrecy of the key K.

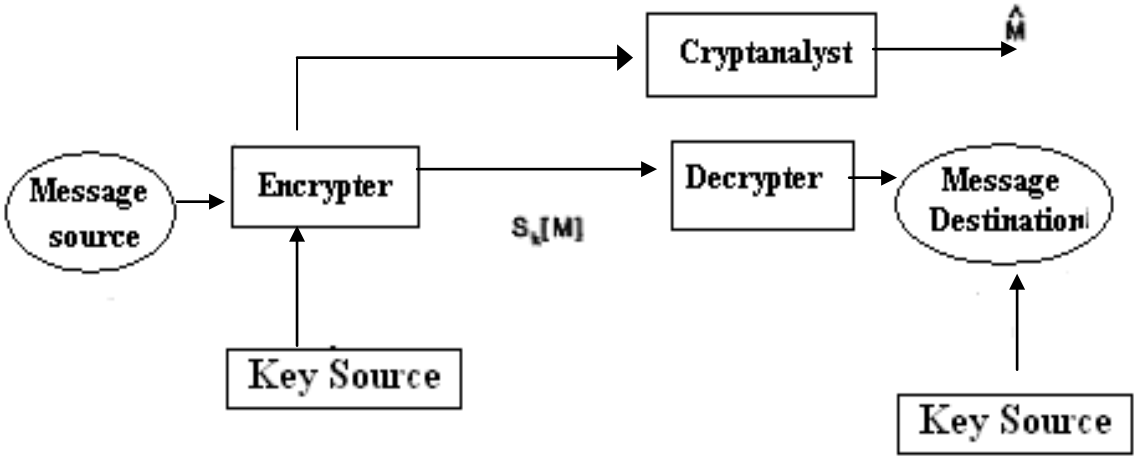


Figure 1.2: The secure communication model.

1.4 Project Modules.

The Project was broken into following work phases/ modules.

- Selection of the Cryptographic Algorithms.
- Study of Advanced Encryption Standards (AES).
- Matlab Implementation of the Advanced Encryption Standard.
- Design Of the speech encryption system.
- C-coding
- Real Time implementation.

1.4.1 Selection of the Cryptographic Algorithms.

The very first phase was the selection of the cryptographic algorithm. We selected advanced encryption standard which is arguably the most secure algorithm in the present day world. The in detail reasoning as to how AES evolved to have the stated status are explained the chapter 3.

1.4.2 Study of Advanced Encryption Standards (AES).

Having selected the cryptographic algorithm the next task at hand was to study and develop adequate understanding. The FIPS 197 and the Rijndael submission by its authors to NIST were the two documents that proved to be very helpful. These documents explain the standard down to smallest details such as mathematical preliminaries. Once the study of the algorithm was completed the next phase was its implementation.

1.4.3 Matlab Design of the Advanced Encryption Standard.

The Matlab design of the algorithm is explained in detail in Chapter 4 of this document. The Matlab implementation of the standard was perhaps the most important aid to understand the functionality of the algorithm. Matlab is a powerful design tool which enabled us to leave the programming worries aside and concentrate on understanding the algorithm. All the test vectors given the FIPS 197 were verified thus proving the correctness of the implementation.

1.4.4 Design of the speech encryption system.

The authors of the Rijndael had specifically stated the suitability of the algorithm for ATM, HDTV, B-ISDN, voice and satellite. Hence AES was put to use to encrypt audio.

Once having implemented the Advanced Encryption Standard in Matlab, the code was put to use to encrypt audio. The computational intensive nature of the algorithm and also the fact that Matlab is essentially a design tool hampered the real time implementation on the said platform. However this was not an intended goal from the

very beginning. The Matlab implementation however helped to judge the feasibility of the project and the transmission of the encrypted speech on the channel.

1.4.5 C-coding

The next important task at hand was the C coding of the AES. The implementation in Matlab had simplified the task as the algo was clear now. The encrypter and decrypted were implemented initially in C++, however the code was slightly modified to convert it to C which was required for the implementation on the DSK 6711. The C++ implementation achieved the desired result on an 800 MHz processor and the real time implementation was now visible as possible.

1.4.6 Real Time implementation.

The final task of the Project was the real time encryption and decryption. Two Platforms were considered in this regard i.e. The DSK6711 and the PC.

A complete secure communication system was realized using the windows based personal computers communicating via the serial port.

Chapter 2

Cryptology.

2.1 Introduction

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. Cryptography has a long and fascinating history. It is a complex art which has been practiced over the ages. The predominant practitioners of the art were those associated with the military, the diplomatic service and government in general. Encryption was used as a tool to protect national secrets and strategies. The proliferation of computers and communications systems in the 1960s brought with it a demand from the private sector for means to protect information in digital form and to provide security services. Beginning with the work of Feistel at IBM in the early 1970s and culminating in 1977 with the adoption as a U.S. Federal Information Processing Standard for encrypting unclassified information, DES, the Data Encryption Standard. Over the centuries, an elaborate set of protocols and mechanisms has been created to deal with information security issues when the information is conveyed by physical documents. Often the objectives of information security cannot solely be achieved through mathematical algorithms and protocols alone, but require procedural techniques and abidance of laws to achieve the desired result.

Conceptually, the way information is recorded has not changed dramatically over time. Whereas information was typically stored and transmitted on paper, much of it

now resides on magnetic media and is transmitted via telecommunications systems, some wireless. What has changed dramatically is the ability to copy and alter information. One can make thousands of identical copies of a piece of information stored electronically and each is indistinguishable from the original. With information on paper, this is much more difficult. What is needed then for a setup where information is mostly stored and transmitted in electronic form is a means to ensure information security which is independent of the physical medium recording or conveying it and such that the objectives of information security rely solely on digital information itself. Information security manifests itself in many ways according to the situation and requirement. Achieving information security in an electronic society requires a vast array of technical and legal skills. There is, however, no guarantee that all of the information security objectives deemed necessary can be adequately met. The technical means is provided through cryptography.

2.2 Cryptographic goals

Of all the information security objectives, the following four form a framework upon which the others are derived:

- Privacy or confidentiality.
- Data integrity.
- Authentication.
- Non-repudiation.

2.2.1 Confidentiality

It is a service used to keep the content of information from all but those authorized to have it. Secrecy is a term synonymous with confidentiality and privacy. There are

numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.

2.2.2 Data integrity

This service addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.

2.2.3 Authentication

It is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).

2.2.4 Non-repudiation

Non-repudiation is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, one entity may authorize the purchase of property by another entity and later deny such authorization was granted. A procedure involving a trusted third party is needed to resolve the dispute.

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities.

2.3 Criterion for Assessing Cryptographic Algorithms.

The cryptographic algorithm comes with their own weaknesses and strength. There cannot be a standard measuring of the ability of the cryptographic tools; however over the years various criterions have been set to asses the cryptographic algorithms. They are as follows.

- Level of security.
- Functionality.
- Methods of operation.
- Performance. Ease of implementation.

2.3.1 Level of security.

This is usually difficult to quantify. Often it is given in terms of the number of operations required (using the best methods currently known) to defeat the intended objective. For example to break the security of a key based algorithm it would be typical to calculate all the possible key combinations. Recently the Data Encryption Standard was cracked in a similar fashion by brute force attack. The level of security of a key based algorithm can be increased by increasing the key size. Typically the level of security is defined by an upper bound on the amount of work necessary to defeat the objective. This is sometimes called the work factor.

2.3.2 Functionality.

Cryptographic algorithms will need to be combined to meet various information security objectives. Which cryptographic algorithms are most effective for a given objective will be determined by the basic properties of the cryptographic algorithms.

2.3.3 Methods of operation.

Cryptographic algorithms, when applied in various ways and with various inputs, will typically exhibit different characteristics; thus, one primitive could provide very different functionality depending on its mode of operation or usage.

2.3.4 Performance.

This refers to the efficiency of a primitive in a particular mode of operation.
(For example, an encryption algorithm may be rated by the number of bits per second which it can encrypt.)

2.3.5 Ease of implementation.

This refers to the difficulty of realizing the primitive in a practical instantiation. This might include the complexity of implementing the primitive in either a software or hardware environment. The relative importance of various criteria is very much dependent on the application and resources available. For example, in an environment

where computing power is limited one may have to trade off a very high level of security for better performance of the system as a whole. Cryptography, over the ages, has been an art practiced by many who have devised ad hoc techniques to meet some of the information security requirements. The last twenty years have been a period of transition as the discipline moved from an art to a science. There are now several international scientific conferences devoted exclusively to cryptography and also an international scientific organization, the International Association for Cryptologic Research (IACR), aimed at fostering research in the area.

2.4 A taxonomy of cryptographic primitives.

Security Primitives can be broadly classified as

- Unkeyed Primitives.
- Symmetric-key Primitives.
- Public-key Primitives.

2.4.1 Unkeyed Primitives.

The Unkeyed primitives are those which donot make use of a key for encrypting the plain text. They can be further classified as

- Arbitrary length hash functions
- One-way permutations

2.4.1.1 Arbitrary length hash functions

A hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values.

2.4.1.2 One-way permutations

Permutations are functions which are often used in various cryptographic constructs. A permutation p on S is a bijection. For example permutation p: S −S is defined as follows:

P (1) = 3; P (2) = 5; P (3) = 4; P (4) = 2; P (5) = 1:

A permutation can be described in various ways. It can be displayed as above or as an array:

P = (1 2 3 4 5)
(3 5 4 2 1)

2.4.2 Symmetric Key Primitives.

They are further classified as.

- Symmetric-key ciphers
- Pseudorandom sequences

2.4.2.1 Symmetric-key ciphers

The cryptographic primitives that make use of the same key for encryption and decryption are known as symmetric key primitives.

Symmetric Key ciphers are further classified as

- Block Ciphers
- Stream Ciphers.

2.4.2.1.1 Block Ciphers

A block cipher is a function which maps n-bit plaintext blocks to n-bit cipher text blocks; n is called the block length. Symmetric-key block ciphers are the most prominent and important elements in many cryptographic systems. Individually, they provide confidentiality. As a fundamental building block, their versatility allows construction of pseudorandom number generators, stream ciphers, MACs, and hash functions. They may furthermore serve as a central component in message authentication techniques, data integrity mechanisms, entity authentication protocols, and (symmetric-key) digital signature schemes. This chapter examines symmetric-key block ciphers, including both general concepts and details of specific algorithms. No block cipher is ideally suited for all applications, even one offering a high level of security. This is a result of inevitable tradeoffs required in practical applications, including those arising from, for example, speed requirements and memory limitations (e.g., code size, data size, cache memory), constraints imposed by implementation platforms (e.g., hardware, software, chip cards), and differing tolerances of applications to properties of various modes of operation. In addition, efficiency must typically be traded off against security. Thus it is beneficial to have a number of candidate ciphers from which to draw.

2.4.2.1.2 Stream Ciphers.

Stream ciphers are an important class of encryption algorithms. They encrypt individual characters (usually binary digits) of a plaintext message one at a time, using an encryption transformation which varies with time. By contrast, block ciphers tend to simultaneously encrypt groups of characters of a plaintext message using a fixed encryption transformation. Stream ciphers are generally faster than block ciphers in hardware, and have less complex hardware circuitry. They are also more appropriate, and in some cases mandatory (e.g., in some telecommunications applications), when buffering is limited or when characters must be individually processed as they are received. Because they have limited or no error propagation, stream ciphers may also be advantageous in situations where transmission errors are highly probable. There is a vast body of theoretical knowledge on stream ciphers, and various design principles for stream ciphers have been proposed and extensively analyzed. However, there are relatively few fully-specified stream cipher algorithms in the open literature. This unfortunate

state of affairs can partially be explained by the fact that most stream ciphers used in practice tend to be proprietary and confidential. By contrast, numerous concrete block cipher proposals have been published, some of which have been standardized or placed in the public domain. Nevertheless, because of their significant advantages, stream ciphers are widely used today, and one can expect increasingly more concrete proposals in the coming years.

2.4.2.2 Pseudorandom sequences

The Pseudorandom sequences are those sequences which are triggered by a key. The encryption and decryption of such a sequence is done through similar key.

2.4.3 Public-key Primitives.

They are further classified as

- Public-key ciphers
- Identification primitives
- Signatures

2.4.3.1 Public-key ciphers

The cryptographic primitives that make use of a different key for encryption and decryption are known as public key ciphers.

2.4.3.2 Identification primitives

An identification or entity authentication technique assures one party (through acquisition of corroborative evidence) of both the identity of a second party involved, and that the second was active at the time the evidence was created or acquired. Typically the only data transmitted is that necessary to identify the communicating parties. The entities are both active in the communication, giving a timeliness guarantee.

2.4.3.3 Signatures

A cryptographic primitive which is fundamental in authentication, authorization, and non repudiation is the digital signature. The purpose of a digital signature is to provide a means for an entity to bind its identity to a piece of information. The

process of signing entails transforming the message and some secret information held by the entity into a tag called

a signature.

2.5 Classes of attacks and security models

Over the years, many different types of attacks on cryptographic primitives and protocols have been identified. The attacks these adversaries can mount may be classified as follows:.

- A passive attack is one where the adversary only monitors the communication channel. A passive attacker only threatens confidentiality of data.
- An active attack is one where the adversary attempts to delete, add, or in some other way alter the transmission on the channel. An active attacker threatens data integrity and authentication as well as confidentiality.

A passive attack can be further subdivided into more specialized attacks for deducing plaintext from cipher text

2.5.1 Attacks on encryption schemes

The objective of the following attacks is to systematically recover plaintext from cipher text, or even more drastically, to deduce the decryption key.

A cipher text-only attack is one where the adversary (or cryptanalyst) tries to deduce the decryption key or plaintext by only observing cipher text. Any encryption scheme vulnerable to this type of attack is considered to be completely insecure.

2.5.1.1 Known-plaintext attack

A known-plaintext attack is one where the adversary has a quantity of plaintext and corresponding cipher text. This type of attack is typically only marginally more difficult to mount.

2.5.1.2 Chosen-plaintext attack

A chosen-plaintext attack is one where the adversary chooses plaintext and is then given corresponding cipher text. Subsequently, the adversary uses any information deduced in order to recover plaintext corresponding to previously unseen cipher text.

2.5.1.3 Adaptive chosen-plaintext attack

An adaptive chosen-plaintext attack is a chosen-plaintext attack wherein the choice of plaintext may depend on the cipher text received from previous requests.

2.5.1.4 Chosen-cipher text attack

A chosen-cipher text attack is one where the adversary selects the ciphertext and is then given the corresponding plaintext. One way to mount such an attack is for the adversary to gain access to the equipment used for decryption (but not the decryption key, which may be securely embedded in the equipment). The objective is then to be able, without access to such equipment, to deduce the plaintext from (different) cipher text.

2.5.1.5 Adaptive chosen-cipher text attack

An adaptive chosen-cipher text attack is a chosen-cipher text attack where the choice of cipher text may depend on the plaintext received from previous requests. Most of these attacks also apply to digital signature schemes and message authentication codes. In this case, the objective of the attacker is to forge messages or Macs.

2.5.2 Models for evaluating security

The security of cryptographic primitives and protocols can be evaluated under several different models. The most practical security metrics are computational, provable, and ad hoc methodology, although the latter is often dangerous. The confidence level in the amount of security provided by a primitive or protocol based on computational or ad hoc security increases with time and investigation of the scheme. Following are the models for evaluating security.

- Unconditional security
- Complexity-theoretic security
- Provable security
- Computational security
- Ad hoc security

2.5.2.1 Unconditional security

The most stringent measure is an information-theoretic measure – whether or not a system has unconditional security. An adversary is assumed to have unlimited computational resources, and the question is whether or not there is enough

information available to defeat the system. Unconditional security for encryption systems is called perfect secrecy. For perfect secrecy, the uncertainty in the plaintext, after observing the cipher text, must be equal to the a priori uncertainty about the plaintext – observation of the cipher text provides no information whatsoever to an adversary. A necessary condition for a symmetric-key encryption scheme to be unconditionally secure is that the key be at least as long as the message.

2.5.2.2 Complexity-theoretic security

An appropriate model of computation is defined and adversaries are modeled as having polynomial computational power. (They mount attacks involving time and space polynomial in the size of appropriate security parameters.) A proof of security relative to the model is then constructed. An objective is to design a cryptographic method based on the weakest assumptions possible anticipating a powerful adversary. Asymptotic analysis and usually also worst-case analysis is used and so care must be exercised to determine when proofs have practical significance. In contrast, polynomial attacks which are feasible under the model might, in practice, still be computationally infeasible. Security analysis of this type, although not of practical value in all cases, may nonetheless pave the way to a better overall understanding of security. Complexity-theoretic analysis is invaluable for formulating fundamental principles and confirming intuition. This is like many other sciences, whose practical techniques are discovered early in the development, well before a theoretical basis and understanding is attained.

2.5.2.3 Provable security

A cryptographic method is said to be provably secure if the difficulty of defeating it can be shown to be essentially as difficult as solving a well-known and supposedly difficult (typically number-theoretic) problem, such as integer factorization or the computation of discrete logarithms. Thus, “provable” here means provable subject to assumptions. This approach is considered by some to be as good a practical analysis technique as exists. Provable security may be considered part of a special sub-class of the larger class of computational security considered next.

2.5.2.4 Computational security

This measure the amount of computational effort required, by the best currently-known methods, to defeat a system; it must be assumed here that the system has been well-studied to determine which attacks are relevant. A proposed technique is said to be computationally secure if the perceived level of computation required to defeat it (using the best attack known) exceeds, by a comfortable margin, the computational resources of the hypothesized adversary. Often methods in this class are related to hard problems but, unlike for provable security, no proof of equivalence is known. Most of the best known public-key and symmetric key schemes in current use are in this class. This class is sometimes also called practical security.

2.5.2.5 Ad hoc security

This approach consists of any variety of convincing arguments that every successful attack requires a resource level (e.g., time and space) greater than the fixed resources of a perceived adversary. Cryptographic primitives and protocols which survive such analysis are said to have heuristic security, with security here typically in the computational sense.

Chapter 3

Evolution of the Advanced Encryption Standard (AES).

3.1 Introduction.

Cryptology is the art and science of making and breaking mathematical schemes that attempt to prevent the disclosure of information to others. It is most often used to conceal information hat could be potentially harmful if it was made publicly available.

The great majority of Cryptology that most people know of comes directly from the ideas of classical Cryptology, where poly alphabetic substitution and other trivial methods are used to try to protect secret messages. Children often use single alphabetic substitution to encipher messages they send to friends, and it should be noted that such encryption can often be broken in a fraction of a second on a modern personal computer.

Modern Cryptology, in contrast with classical Cryptology, is often much more mathematically involved. It is modern Cryptology that plays the greatest role in our everyday lives, and it is modern Cryptology that is most often used on computers. Cryptology is needed in today's world for many reasons. It is often used for secure online transactions over the Internet, banking, digital authentication and verification, electronic mail, online privacy, and many other situations where private communications are required.

Modern Cryptology has its roots in mathematics because mathematics makes it easier to understand and verify that a cryptosystem is indeed secure. These mathematical problems are easy to solve given some inputs, but it is nearly impossible to recover the original inputs if given the answer. Cryptology as a whole consists of two main areas of study: cryptography and cryptanalysis. Cryptography is simply defined as the art of creating and using cryptosystems, where a cryptosystem is any way of disguising messages so that only certain people can see through the veil of secrecy. Cryptanalysis, in turn, is the art of breaking cryptosystems, or in other words seeing through the veil of secrecy even when one is not the person to whom the message was intended.

Both cryptography and cryptanalysis are intense areas of study in today's world, as the demand for communications security reaches unprecedented levels. Just as modern technology has increased the strength and complexity of cryptosystems, it has also allowed for the creation of a wider range of increasingly sophisticated attacks against them.

3.2 Need for encryption

Streams of information flow around the world. More and more networks are being set up both locally and globally. The channels through which your information passes are increasingly being taken out of control. Transmission actually takes place via invisible high-tech channels, and although this is very efficient, it is not secure. Particularly if your data is sensitive and much depends on its content.

What can actually happen to your information while en route? You have probably asked yourself that question many times and probably imagined something worrying. You don‘t want any unauthorized persons reading, viewing or listening to confidential information, manipulating data, or helping themselves to the traffic on the data

highway.

Encryption is an elegant way of solving security problems against unauthorized access, even before they have begun their travels. Flexibility and modular structure enable solutions that are completely tailor-made to your special needs as a public authority or company. Cryptography is the science of encrypting and decrypting data for security reasons. The above-mentioned security services can now be performed by implementing cryptographic mechanisms. These mechanisms use cryptographic algorithms. Different solutions use symmetric encryption schemes (secret key cryptography) as well as asymmetric encryption schemes (public key cryptography). These solutions are based upon highest quality proprietary symmetric cipher algorithms and well known public key mechanisms like RSA, Diffie-Hellman, elliptic curves etc.

3.3 Necessity of an Encryption Standard.

In 1972, the National Institute of Standards and Technology (called the National Bureau of Standards at the time) decided that a strong cryptographic algorithm was needed to protect non-classified information. The algorithm was required to be cheap, widely available, and very secure. NIST envisioned something that would be available to the general public and could be used in a wide variety of applications. So they asked for public proposals for such an algorithm

3.4 The Predecessor Of AES ; Data Encryption Standard (DES).

In 1974 IBM submitted the Lucifer algorithm, which appeared to meet most of NIST's design requirements. NIST enlisted the help of the National Security Agency to evaluate the security of Lucifer. At the time many people distrusted the NSA due to

their extremely secretive activities, so there was initially a certain degree of skepticism regarding the analysis of Lucifer. One of the greatest worries was that the key length, originally 128 bits, was reduced to just 56 bits, weakening it significantly. The NSA was also accused of changing the algorithm to plant a "back door" in it that would allow agents to decrypt any information without having to know the encryption key. But these fears proved unjustified and no such back door has ever been found. The modified Lucifer algorithm was adopted by NIST as a federal standard on November 23, 1976. Its name was changed to the Data Encryption Standard (DES). The algorithm specification was published in January 1977, and with the official backing of the government it became a very widely employed algorithm in a short amount of time. For many years the well-known cipher DES (Digital Encryption Standard) was the cryptographic standard for unclassified use within the USA. DES has been applied in banks as well as in many software- and hardware products for more than 20 years. One could say that this algorithm is the most widespread cipher at all. Some experts warned that some trapdoor might be built in because the secret service NSA was consulted during the design. Such a trapdoor was never found, although significant theoretical weaknesses which could hardly be exploited in practice were identified shortly after DES was adopted. DES encrypts and decrypts data in 64-bit blocks, using a 64-bit key .It takes a 64-bit block of plaintext as input and outputs a 64-bit block of ciphertext. Since it always operates on blocks of equal size and it uses both permutations and substitutions in the algorithm, DES is both a block cipher and a product cipher. DES has 16 rounds, meaning the main algorithm is repeated 16 times to produce the ciphertext. It has been found that the number of rounds is exponentially proportional to the amount of time required to find a key using a brute-force attack. So as the number of rounds increases, the security of the

algorithm increases exponentially. Although the input key for DES is 64 bits long, the actual key used by DES is only 56 bits in length. The least significant (right-most) bit in each byte is a parity bit, and should be set so that there are always an odd number of 1s in every byte. These parity bits are ignored, so only the seven most significant bits of each byte are used, resulting in a key length of 56 bits.

3.5 The Cracking of DES

The weak point of DES was not its design, but its key size of only 56 bit. This corresponds to about 72,000,000,000,000,000 possible keys. In the seventies (when DES was introduced), this was an astronomical number. Meanwhile hardware has become very fast. In summer 1998 an organization named EFF built and demonstrated a special computer, Deep Crack, that could decrypt a DES-enciphered text within an average time of 4.5 days. Crypto experts were not surprised. Unfortunately, over time various shortcut attacks were found that could significantly reduce the amount of time needed to find a DES key by brute force. And as computers became progressively faster and more powerful, it was recognized that a 56-bit key was simply not large enough for high security applications. As a result of these serious flaws, NIST abandoned their official endorsement of DES in 1997 and began work on a replacement, to be called the Advanced Encryption Standard (AES). As an interim measure a variant of the DES algorithm known as triple DES or 3 DES was put in place.

3.6 Triple DES (3DES)

3DES is a threefold DES encryption in which two 56-bit keys are applied. That means a practical security of 112 bits in strength which, as far we know today, should be reasonably secure. Triple DES was the answer to many of the shortcomings of

DES. Since it is based on the DES algorithm, it is very easy to modify existing software to use Triple DES. It also has the advantage of proven reliability and a longer key length that eliminates many of the shortcut attacks that can be used to reduce the amount of time it takes to break DES. However, even this more powerful version of DES was not strong enough to protect data for very much longer. The DES algorithm itself has become obsolete and is in need of replacement. To this end the National Institute of Standards and Technology (NIST) held a competition to develop the Advanced Encryption Standard (AES) as a replacement for DES. Triple DES has been endorsed by NIST as a temporary standard to be used until the AES was finished.Triple DES is simply another mode of DES operation. It takes three 64-bit keys, for an overall key length of 192 bits. In Stealth, you simply type in the entire 192-bit (24 character) key rather than entering each of the three keys individually. The Triple DES then breaks the user provided key into three subkeys, padding the keys if necessary so they are each 64 bits long. The procedure for encryption is exactly the same as regular DES, but it is repeated three times. Hence the name Triple DES. The data is encrypted with the first key, decrypted with the second key, and finally encrypted again with the third key.

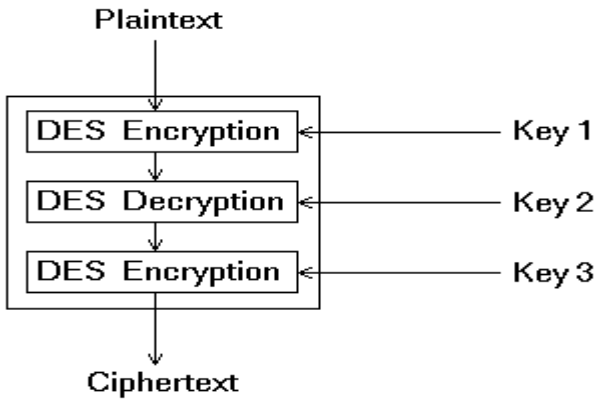


Figure 3.1 Block diagram of 3-DES

3.7 The Competition for AES.

DES was now outdated. Even 3DES did not fulfill the requirements of a modern algorithm - it must be suited for implementation in software as well as in hardware. (DES is functionally only hardware-friendly.) – The need for a new standard was imminent. It was clear that the new standard must have variable key and block sizes and - it must be able to resist all known crypt analytical attacks, even if these can not be used in practice yet.

In the beginning of 1997 NIST formulated the call for a new algorithm with the name "AES" (Advanced Encryption Standard). It was clear that this process would take a long time: Still more than the design of a cipher, its analysis is an extraordinary difficult task, except if the cipher is obviously weak. To say it frankly, this is an unsolvable task. One can not prove that some algorithm is secure (with a single exception of almost no practical interest); one can only show that known attack methods do not work. Nevertheless, cryptographers have to take this challenge. Since cryptanalysis has been strongly developed, particularly in the past decade, we have a rich experience in cipher evaluation today.

3.7.1 AES Requirements

These requirements where formulated by NIST after public discussion, among others during a workshop at 15 April 1997. Here are only some criteria: It must be a symmetric cipher (the same key is used for encryption and decryption), more precisely a block cipher. (The public cryptanalysis of so-called stream ciphers seems not to

be so well developed). The block size must be at least 128 bits; keys of 128, 192 and 256 bits must be applicable. AES must be well suited to different purposes, thus is

must be easy to implement in both hard and software and perform well in both cases. AES must be able to resist all known methods of cryptanalysis. Especially, AES must be resistant against timing and power attacks. For the use inside smartcards in particular, it must be able to work with limited resource availability (short code, small memory). There must be no patent right on the algorithm. It must be free to use for everybody. A remark concerning the demanded key and block sizes is appropriate here. During a public discussion in Germany some years ago, a legal expert declared that trying out all possible keys is only a question of “dedication
and industriousness”. The example of a 256-bit key shows exactly how “dedication” is really needed: Suppose, for instance, that you would store all 2256 possible keys and only one atom were used for each key (currently nobody knows how to do that!). The required memory would have a mass of about 1047 grams. This is about 1014 times the weight of a typical star. As Bruce Schneier has already mentioned, in such cases the storage medium would create a black hole and thus never let out any information. For 128-bit keys this memory would only have a mass of about 300 tons. It would still store the incredible amount of 1024 TB. We did not speak yet about the CPU to perform the necessary calculations. However, for AES such thought experiments are no motivation. More realistic dangers are first that the still hypothetical quantum computers could reach a new quality in parallelizing attacks. Secondly, it is always possible that AES could be attacked more elegantly than with brute force (i.e., trying all possible keys). In this case we need a large safety margin. By the way, 3DES could never be an AES candidate because of its “short” key size of 112 bits.

3.7.2 Creation of the Algorithm

Experts have said it over and over again: Only a publicly analyzed algorithm can be secure. Ciphers with secret designs are almost always weak. There is an intrinsic example for this: The algorithms A3 and A5 used in GSM mobile phones were disclosed not long after their application. A3 is responsible for authentication. After only one day of analysis it was broken: Mobile phones can be cloned in practice now (i.e., you can create your own SIM card with a false number and use it charging someone else). Even if an algorithm is known it should be clear why it was designed just this way. If this is not clear, users will distrust it. The best example is DES: The modification of the so-called “S-boxes” (presumably by NSA) caused the suspicion that some trapdoor was built in. Some design principles were published years later, but it was too late. However, the AES election procedure was really open: Anybody could propose a cipher, the best cryptologists in the world analyzed it. And the design principles were also published.

3.7.3 The Selection Procedure

The NIST received 15 proposals for algorithms. After thorough investigation, five candidates remained. The best cryptanalysts of the world dealt with them. Now a pat situation arose: All five algorithms were excellent and were also very hard to compare. Each of them could have

easily become the new standard; none of them had any significant weak point. Each of them had some advantages over the others - but what properties should be the most important? During the Third AES conference Don B. Johnson (Certicom) asked in two lectures: Must there be a best algorithm at all? Modern software anyway

implements some normed crypto API and offers several algorithms in parallel. The five AES candidates are small enough to be all contained in one product. Of course, the situation for hardware is not so convenient. So it would be prudent to prefer different algorithms for different purposes in hardware (smartcards, online enciphering, enciphering,) and to offer all five in software. Such a flexibility would have more advantages than disadvantages. Products that are fixed on some algorithm would become suddenly insecure if a weakness of these algorithms is discovered in spite of all expectations. Remember the use of DES in banks? It was applied for about 20 years. The switch to 3DES took several years and consumed huge amounts of money. The Adoption of Rijndael Ultimately, NIST decided differently from what most people expected - only one algorithm was the winner, Rijndael, the Belgian algorithm developed by Joan Daemen and Vincent Rijmen. The official reasons for this decision (which do not convince everyone) are as follows: -

-Should a weakness of Rijndael be discovered, larger key sizes will give more security.

- In the worst case, 3DES could be an alternative for some time (3DES will remain secure in the near future).

- It is cheaper to implement one single algorithm (this argument is certainly only true for hardware - R.W.)

- Developers of similar algorithms could claim patent rights. If only one algorithm is selected, costs will be smaller.

The selection process was open and fair. Not long ago in the USA cryptographic algorithms were still classified as “munitions.” Now a cipher developed in Belgium and evaluated internationally will become a fundament of national security in the

States, although not exclusively, since the US Government will apply it for “sensitive, not classified” information only (and similarly the NSA). But in practice it will dominate - in business, in good crypto products for private use etc. NIST expects AES to be the base for enciphering for the next 20 years or more. There is no reason yet to doubt these assertions.

3.7.4 A Chronology of the NIST AES Initiative

2.1.97: Call for algorithms, accepting proposals until 12.9.97.

5.4.97: During a public AES workshop, the detailed requirements are formulated.

20.8.98: First AES Conference. NIST announces the receipt of 15 algorithms, sent by cryptographers from all over the world. The public evaluation starts.

March 99: Second AES conference. Discussion of obtained results. 28 publications had been submitted before and were put on the NIST homepage to make conference discussions more effective.

15.4.99: End of public evaluation of all candidates. Five candidates (MARS, RC6, Rijndael, Serpent, Twofish) are in the next round. Further work will be concentrated on these algorithms from now.

13./14.4.00: Third AES Conference. The analysis of the final five candidates are presented and discussed.

15.5.00: End of public discussion.

2.10.00: The "winner," Rijndael, is announced.

November 2000: The FIPS standard is published as a draft. Public comments are possible.

February 2001: End of the public discussion of the standard.

April-June 2001: Confirmation of the FIPS standard.

3.7.5 Submitted algorithms

Round 1:

CAST-256: Entrust Technologies, Inc.
CRYPTON: Future Systems, Inc.
DEAL: Richard Outerbridge, Lars Knudsen
DFC: CNRS Centre National pour la Recherche
Scientifique - Ecole Normale Superieure
E2: NTT - Nippon Telegraph and Telephone Corporation
FROG: TecApro Internacional S.A.
HPC: Rich Schroeppel
LOKI97: Lawrie Brown, Josef Pieprzyk, Jennifer Seberry
MAGENTA: Deutsche Telekom AG
MARS: IBM
RC6: RSA Laboratories
RIJNDAEL: Joan Daemen, Vincent Rijmen
SAFER+: Cylink Corporation
SERPENT: Ross Anderson, Eli Biham, Lars Knudsen
I: Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson

Round 2 finalists:

MARS
RC6
Rijndael
Serpent
Twofish

Selected after Round 3:

Rijndael

Chapter 4

The Advanced Encryption Standard

4.1 Introduction

AES is based on the block cipher Rijndael and became the designated successor of the Data Encryption Standard (DES) which has been implemented in a tremendous number of cryptographic modules worldwide since 1977.

4.2 Finite Field Arithmetics

The following section introduces the different representation forms of a byte and discusses the basic arithmetics of finite fields. A finite field, also called a Galois Field , is a field with only finitely many elements. The finite field GF(28) e. g. consists of the 28 = 256 di_erent numbers (0 . . . 255) represented by one byte (8 bits). Special xor- and modulo-operations, explained in detail in the following sections, make sure that the sum and the product of two finite field elements remain within the range of the original finite field.

4.3 Polynomial Addition

Usually two polynomials are added by adding the coefficients of like powers of x according to Figure 1.

$$(x^6+x^4+x^2+x+1)+(x^7+x^5+x+1)$$

$$x^6 \quad +x^4 \quad +x^2+x+1+$$
$$x^7 \quad +x^5 \quad +x+1$$

$$x^7+x^6+x^5+x^4 \quad +x^2+2x+2$$

Figure 4.1 Classical polynomial addition

Since this might lead to some coefficients of the resulting polynomial not being 0 or 1 (e. g. 2x and 2 in Figure 1), this ”classical” sum does not represent a byte (i. e. an element of the original finite field).In order to make sure that the resulting polynomial has only binary coefficients, the xor (exclusive or) operation depicted in Table 1 is used for the addition. Since the xor-”sum” of two 1’s is not 2 but 0 (1 xor 1 = 0), no 2-coe_cient can appear.

x	y	x xor y
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: xor operation

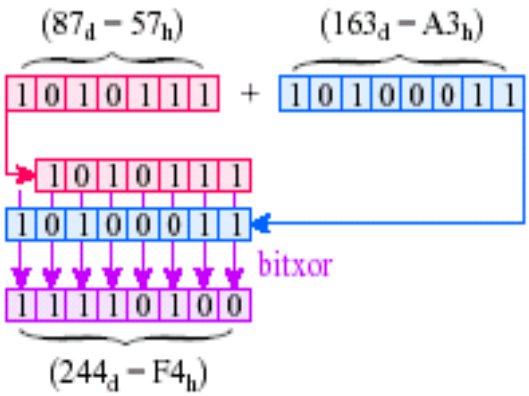


Figure 4.2 Binary polynomial addition

The resulting byte 244d = F4h = 11110100b = x7 + x6 + x5 + x4 + x2 (6)

directly corresponds to the polynomial of Figure 1, if the ”non-binary” terms 2x and 2 are omitted there. The bit-wise xor operation bitxor is a build-in function of Matlab and is used throughout AES, whenever two bytes are added:

bitxor (87, 163)

ans =244

4.4 Polynomial Multiplication

Two polynomials are multiplied by multiplying each summand of the first polynomial by (every summand of) the second polynomial and adding the coefficients of like powers

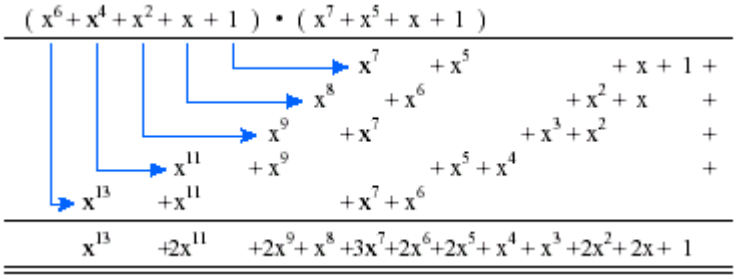


Figure 4.3 ”Classical” polynomial multiplication

Once again, some coefficients of the resulting polynomial in Figure 5.3 are 2 or even 3 and have to be treated differently. The generalization of the xor-concept would now omit every power having an even coefficient and reduce every odd coefficient to 1, leading to a polynomial of x^13 + x^8 + x^7 + x^4 + x^3 + 1.On the bit level (see Figure 4.4) the same result is achieved by shifting the second byte one bit to the left for every bit in the first byte. If a bit in the first byte is 0, a 0-byte is used instead of the second byte. Finally all corresponding bits are xor’ed.

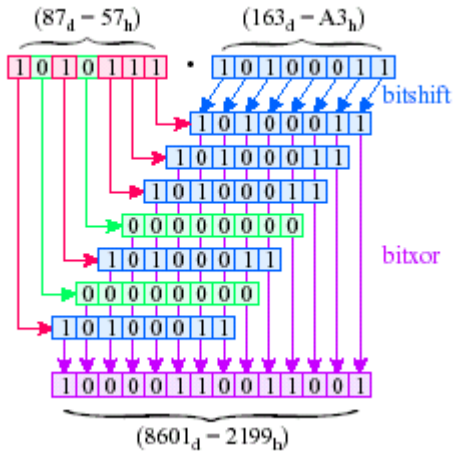


Figure 4.4 Binary polynomial multiplication

Unfortunately the resulting polynomial (7) has a degree greater than 7, can therefore not be expressed in one byte (i. e. it is not a GF(2^8) element) and has to be transformed back into the ”byte range” by the modulo division described in the next section.

4.5 Polynomial Division

The manual algorithm to divide two polynomials is depicted in Figure 4.5.

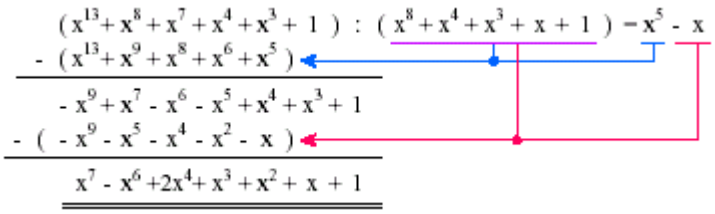


Figure 4.5 Classical polynomial division

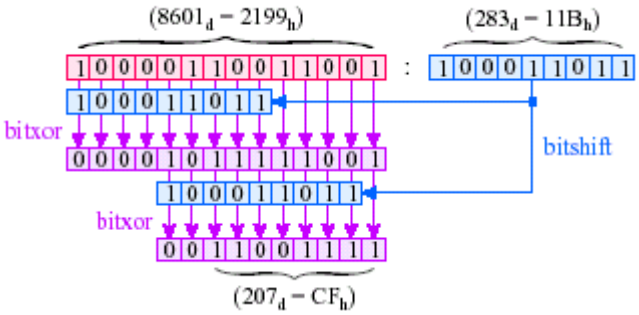
The greatest power of the numerator (x13) is divided by the greatest power of the denominator (x8) yielding the first resulting term (x5). This term is multiplied by the complete denominator (! x13 + x9 + x8 + x6 + x5) and subtracted from the numerator, resulting in a new numerator (-x9 +x7 -x6 -x5 +x4 +x3 +1). This procedure is repeated until the greatest power of the new numerator has become less than the greatest power of the

denominator. The final numerator (x7 - x6 + 2x4 + x3 + x2 + x + 1) is the remainder of this modulo operation. Applying the ”generalized xor-rules” (even coefficients ! 0, odd coefficients ! 1) to the remainder leaves the desired byte-conform polynomial: x7 + x6 + x3 + x2 + x + 1 (8) The bit level operations illustrated in Figure 6 achieve the same result by bit-wise shift and xor operations: The denominator is shifted to the left until its most significant bit (MSB) matches the MSB of the numerator. The subtraction is then performed via xor, resulting in a new, smaller numerator. The shifting and xor-ing is repeated, until the resulting numerator (the remainder) fits into one byte.

(8601 = 2199) d h

(207 = CF) d h

(283 = 11B) d h



:

Figure 4.6 Binary polynomial division

4.7 Algorithm Specification

In the AES algorithm**,** the length of the input block, the output block and the State (which is formed by breaking the speech in the blocks) is 128 bits. For the AES algorithm**,** the length of the Cipher Key, *K*, is variable and can be 128, 192, or 256 bits. For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key size. The only Key-Block-Round combinations that conform to this standard are given in Fig.

	Key Length *(Nk words)*	**Block Size** *(Nb words)*	**Number of Rounds** *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Key-Block-Round Combinations.

For both its Cipher and Inverse Cipher, the AES algorithm uses a round function that is composed of four different byte-oriented transformations: 1) byte substitution using a substitution table (S-box), 2) shifting rows of the State array by different offsets, 3)

mixing the data within each column of the State array, and 4) adding a Round Key to the State.

4.7.1 Cipher

At the start of the Cipher, the input is copied to the State array. After an initial Round Key addition, the State array is transformed by implementing a

round function 10, 12, or 14 times depending on the key length. The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine. The individual transformations involved in the algorithm are as follows: -

1) SubBytes()

2) ShiftRows(),

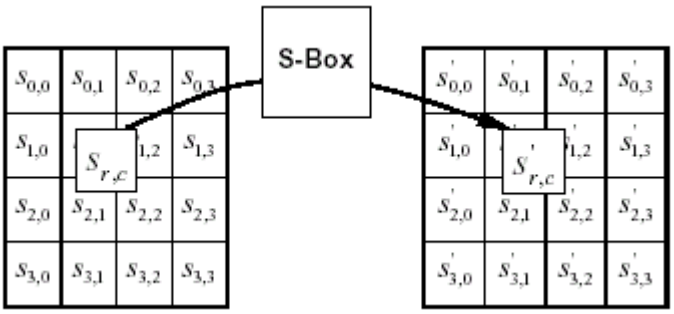
3) MixColumns(),

4) AddRoundKey()

The final round does not include the **MixColumns()** transformation.

4.7.2 SubBytes()Transformation

The **SubBytes()** transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box). The S-Box is formed by taking the multiplicative inverse in the finite field and then applying the affine transformation in the finite field GF (2^8) the element {00} is mapped to itself.



		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 4.7 S-box: substitution values for the byte xy (in hexadecimal format).

4.7.3 ShiftRows() Transformation

In the ShiftRows() transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row is not shifted, row 1 is shifted over 1 byte, row 2 over 2 bytes and row 3 over 3 bytes. This has the effect of moving bytes to “lower” positions in the row while the “lowest” bytes wrap around into the “top” of the row.

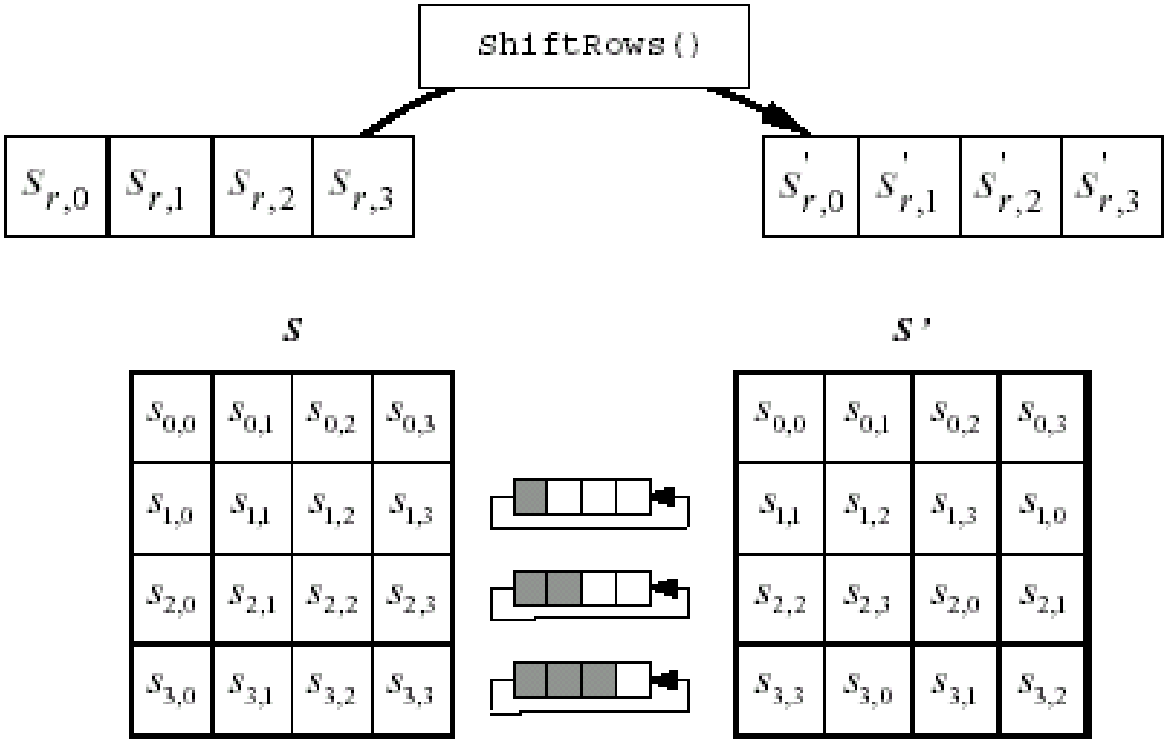


Figure 4.8 ShiftRows() cyclically shifts the last three rows in the State.

4.7.4 MixColumns() Transformation

The function Mixcolumns multiplies each column of the state with the irreducible polynomial given by the following matrix.

x= [2 3 1 1

1 2 3 1

1 1 2 3

3 1 1 2];

The multiplication in the finite field is different from the ordinary multiplication, this is achieved by the function polymult( ) as shown in the figure

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

.

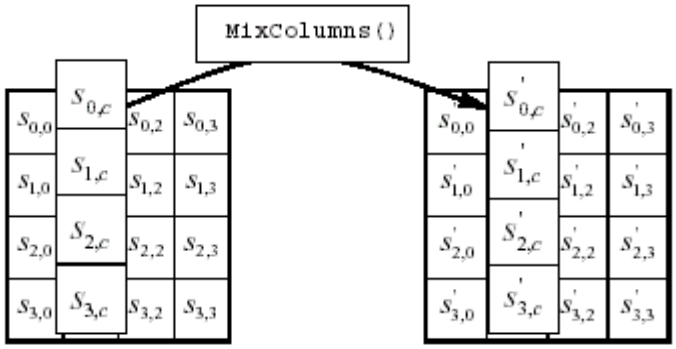


Figure 4.9 MixColumns()operates on the State column-by-column.

4.7.5 AddRoundKey() Transformation

In the **AddRoundKey()** transformation, a Round Key is added to the State by a simple bitwise exclusive or (XOR) operation. The function receives at its input the state, the round number and the array containing the expanded keys. With the information of the round number it extracts the round key for that particular round. Next it performs the bit xor of the state and the round key. This function is called iteratively by the function cipher

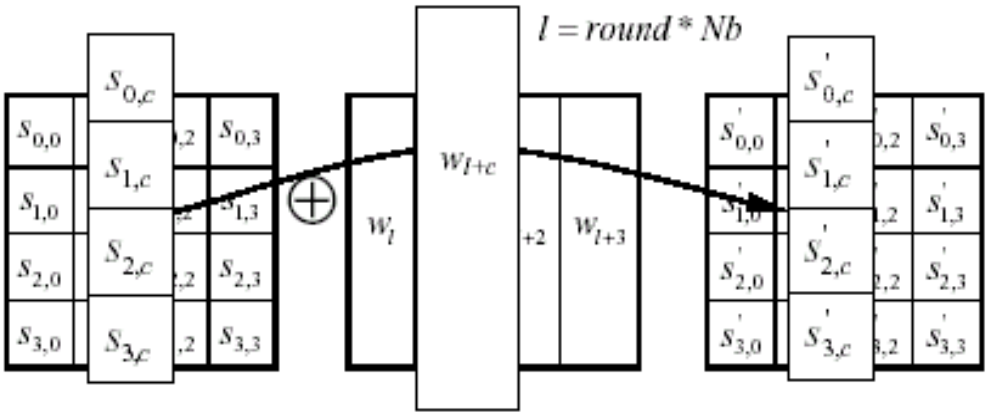


Figure 4.10 Add round key transformation

4.8 Inverse Cipher

The Cipher transformations can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher are

- InvShiftRows(),
- InvSubBytes()
- InvMixColumns(),
- AddRoundKey()

4.8.1 InvShiftRows() Transformation

In Inverse Shiftrows, the rows of the State are cyclically shifted over different offsets. Row 0 is not shifted; Row 1 is shifted over 1 byte, row 2 over 2 bytes and row 3 over 3 bytes. In Matlab this is achieved by a very simple approach of matrix re assignment.

.

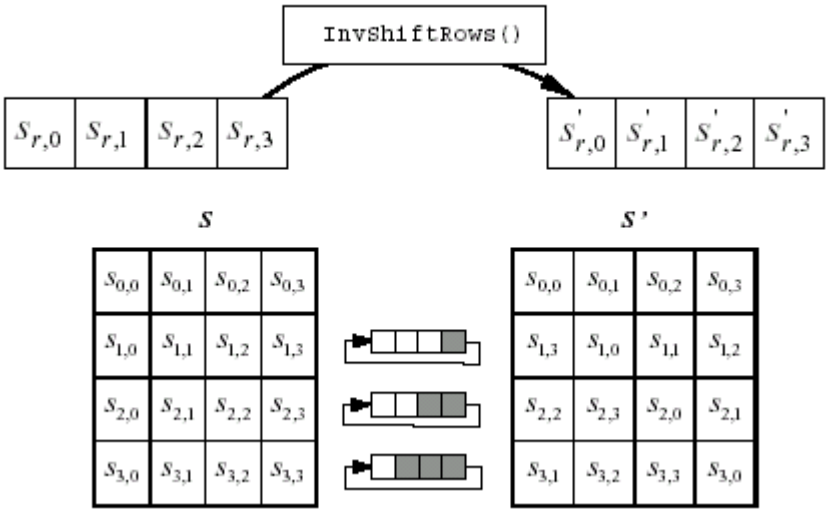


Figure 4.11 InvShiftRows()cyclically shifts the last three rows in the State.

4.8.2 InvSubBytes() Transformation

The variable Sbox contains the Sbox used in Subbytes. The S-Box is formed by taking the multiplicative inverse in the finite field and then applying the affine transformation in the finite field GF (2^8) the element {00} is mapped to itself. The Matlab function Inverse Subbytes uses the code book method and uses an already stored Sbox hence increasing the execution speed. This function accepts the state (which is formed by breaking the speech in the blocks of 128 bit). The function has an already stored Sbox through which the substitution is done.

4.8.3 InvMixColumns() Transformation

The function Inverse Mixcolumns multiplies each column of the state with the irreducible polynomial given by the following matrix.

```
x= [2 3 1 1
    1 2 3 1
    1 1 2 3
    3 1 1 2];
```

The multiplication in the finite field is different from the ordinary multiplication, this is achieved by the function polymult( ) as shown in the figure.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Figure 4.12 Matrix multiplication in Inverse Mix columns

4.8.4 Inverse of the AddRoundKey() Transformation

AddRoundKey, is its own inverse, since it only involves an application of the XOR operation.

Chapter 5

Design Of the project

5.1 Introduction

The actual implementation of AES based encryption system required an implementation of the standard and the verification of the test vectors given in the FIPS 197.The software we chose for the design simulation and testing of our project is MATLAB. This chapter discusses a Matlab implementation speech encryption system based on the Advanced Encryption Standard. AES is based on the block cipher Rijndael and became the designated successor of the Data Encryption Standard (DES) which has been implemented in a tremendous number of cryptographic modules worldwide since 1977. Matlab is a matrix-oriented programming language, perfectly suited for the matrix-based data structure of AES. This implementation is fully operational and executes a speech encryption system based on the advanced encryption standard. The main optimization parameter of this implementation has been execution speed. The shifting and substitution functions have not been coded algorithmically step-by-step instead simple predefined lookup tables have been used. Furthermore, in order to minimize the number of if-then-else-conditions, a key length of 128 bits (16 bytes) has been implemented only; the extension to 24 or 32 bytes key lengths, as defined in, can easily be realized by altering the corresponding constants. The test vectors given in the FIPS 197 have been encrypted and decrypted with successful results. The speech encryption based on AES has been achieved in this implementation. Speech is recorded as wave files using the sound card. It is sampled

at the Nyquist rate and saved as an array. This array is then mu-law encoded. The sampled and mu-law encoded speech is broken up in

packets of 128 bits. Each packet is then reshaped to give it the form of a 4 x 4 array that is the standard form of the AES state. It is then encrypted using the cipher. The encrypted speech is then deciphered. The study of the channel effects on the encrypted speech is underway which would ascertain the feasibility of the wire line or wireless transmission of the ciphered data.

5.2 MATLAB the Language of Technical Computing:

The choice of MATLAB as a test tool was quite obvious. No other software gives you as much of flexibility and enhanced features as are given in MATLAB. The language is being used for design testing and simulation in more than 100 fields of sciences and engineering, which include signal processing, DSP, communications, artificial intelligence, fuzzy logic, neural networks, control system design, data acquisition, image processing, power systems, optimization, spline, statistics, symbolic mathematics, 3d Calculus, system design, system identification, wavelets, financial derivatives and time series, filter design etc. No other software can be used for such an extensive design and testing of communication or signal processing designs. We used the software for communication based system design and simulation.

5.3 Intended Goals of Matlab implementation:

To analyze the working of an AES based encryption system we focused on following aspects.

- Implementation of the AES, encryption and decryption.

- Verification of the test vectors given in the standard.
- Study the suitability of AES for speech and image encryption.

The AES based encrypter and decrypter was programmed in MATLAB and the subsequent testing was carried out. The following sections illustrate the MATLAB implementation of the AES. The main optimization parameter of this implementation has been execution speed. The shifting and substitution functions have not been coded algorithmically step-by-step instead simple predefined lookup tables have been used. Furthermore, in order to minimize the number of if-then-else-conditions, a key length of 128 bits (16 bytes) has been implemented only; the extension to 24 or 32 bytes key lengths, as defined in, can easily be realized by altering the corresponding constants. The test vectors given in the FIPS 197 have been encrypted and decrypted with successful results.

5.4 Modular Description of the Matlab Implementation.

The Matlab implementation can be described to be in three distinct modules

- **Encrypter**
- **Decrypter**
- **The key generator.**

A description of the modules and the member functions is given as follows.

5.4.1 Encrypt.

The function encrypt performs the encryption of the 128 bit block given to it as the input. The function also receives the encryption key.

This function calls the following member function.

- Subbytes ( )
- Shiftrows ( )

- Mixcolumns ( )
- Addroundkey ( )

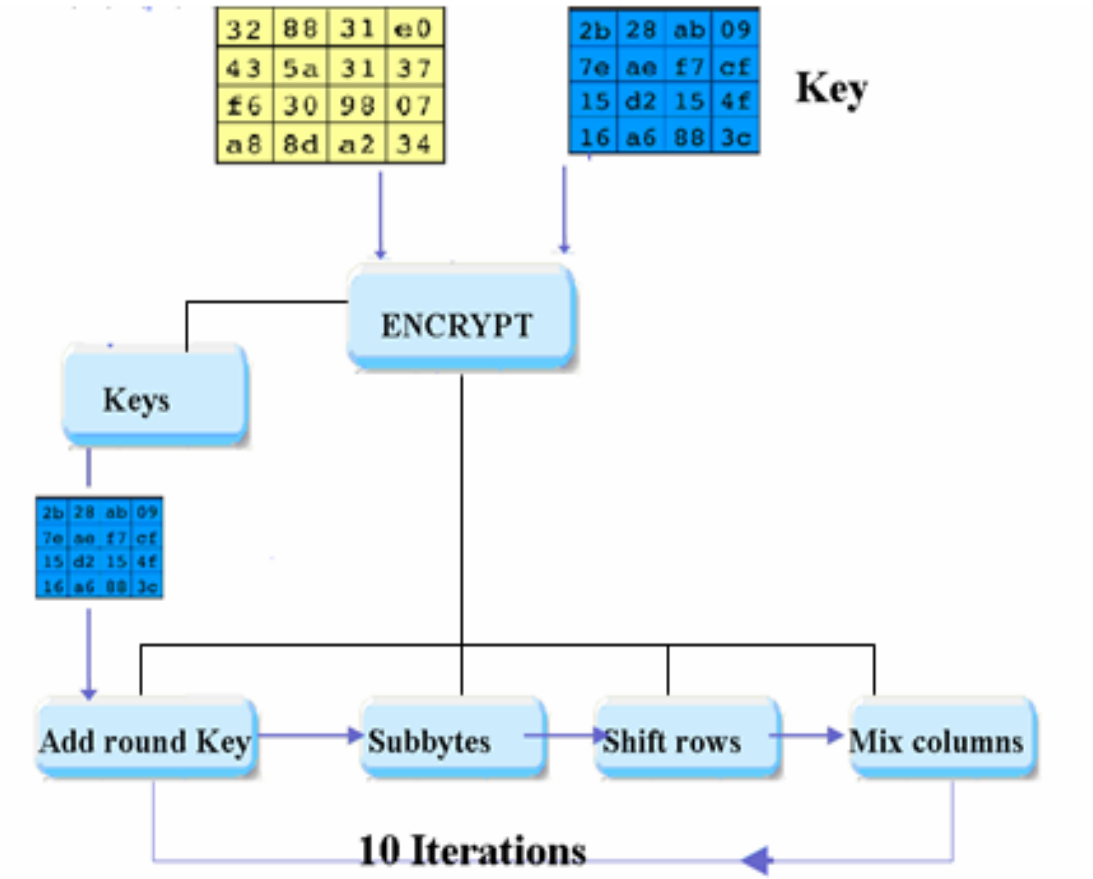


Figure 5.1 The Flow Diagram of Operations

5.4.1.1 Subbytes ( )

The variable Sbox contains the Sbox used in Subbytes. The S-Box is formed by taking the multiplicative inverse in the finite field and then applying the affine transformation in the finite field GF (2^8) the element {00} is mapped to itself. The Matlab function Subbytes uses the code book method and uses an already stored Sbox hence increasing the execution speed. This function accepts the state (which is

formed by breaking the speech in the blocks of 128 bit). The function has an already stored Sbox through which the substitution is done.

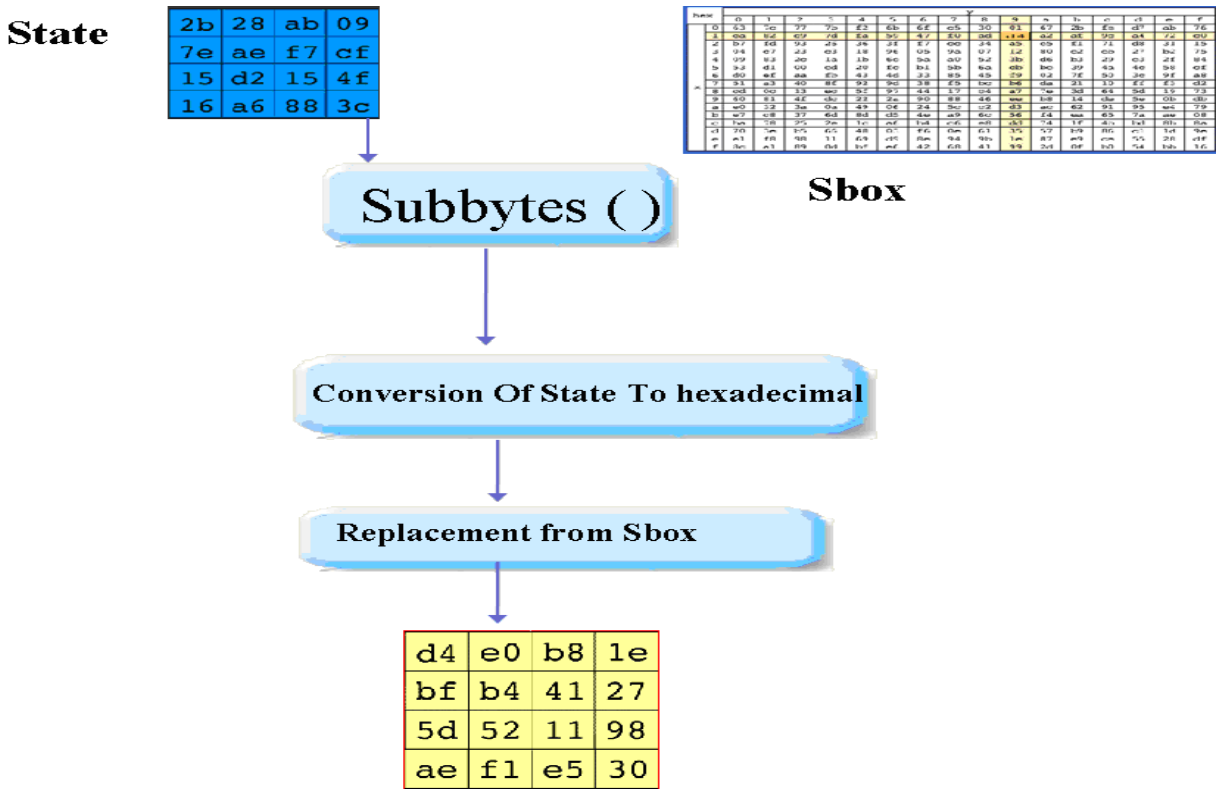


Figure 5.2 Flow Diagram of Subbytes

The Matlab code for Subbytes is given as under.

```
[m,n]=size(state);
 z=dec2hex(state,2);

  for i=1:(m*n)
        state_s(i)=sbox(hex2dec(z(i,1))+1,hex2dec(z(i,2))+1);
 end
  state_s=reshape(state_s,m,n)';
  state_sd=dec2hex( state_s);
```

5.4.1.2 Shiftrows.

In Shiftrows, the rows of the State are cyclically shifted over different offsets. Row 0 is not shifted; Row 1 is shifted over 1 byte, row 2 over 2 bytes and row 3 over 3 bytes. In Matlab this is achieved by a very simple approach of matrix re assignment. The code for this reassignment is as follows.

```
function state_s=shiftrows(state)

a=reshape(state,4,4)';

state_s=[a(1,1) a(1,2) a(1,3) a(1,4)
         a(2,2) a(2,3) a(2,4) a(2,1)
         a(3,3) a(3,4) a(3,1) a(3,2)
         a(4,4) a(4,1) a(4,2) a(4,3)];
```

Figure 5.3 The Matlab Function Shift rows.

5.4.1.3 Mixcolumns.

The function Mixcolumns multiplies each column of the state with the irreducible polynomial given by the following matrix.

```
x= [2 3 1 1
    1 2 3 1
    1 1 2 3
    3 1 1 2];
```

The multiplication in the finite field is different from the ordinary multiplication, this is achieved by the function polymult( ) as shown in the figure.

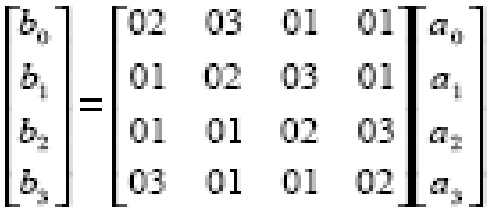


Figure 5.4 Multiplication in Mixcolumns

The code for the function Mixcolumns is given as

```
function state_m=mixcolumns(state)

x=[2 3 1 1
  1 2 3 1
  1 1 2 3
  3 1 1 2];
mod_pol = bin2dec ('100011011');
for i_col_state = 1 : 4
          for i_row_state = 1 : 4
            temp_state = 0;
                 for i_inner = 1 : 4
                        temp_prod = polymult (x(i_row_state, i_inner),state(i_inner,
i_col_state),mod_pol);
          temp_state = bitxor (temp_state, temp_prod);
           end
             state_m(i_row_state, i_col_state) = temp_state;
         end
   end
```

Figure 5.5 Matlab Code for Mixcolumns.

5.4.1.4 Addroundkey.

The function addroundkey performs the bitwise xor of the state and the round key. The function receives at its input the state, the round number and the array

containing the expanded keys. With the information of the round number it extracts the round key for that particular round. Next it performs the bit xor of the state and the round key. This function is called iteratively by the function cipher.

5.4.2 Decrypt.

The function decrypt performs the decryption of the 128 bit encrypted block given to it as the input. The function also receives the encryption key.

This function calls the following member function.

- Inverse Subbytes ( )
- Inverse Shiftrows ( )
- Inverse Mixcolumns ( )
- Addroundkey ( )

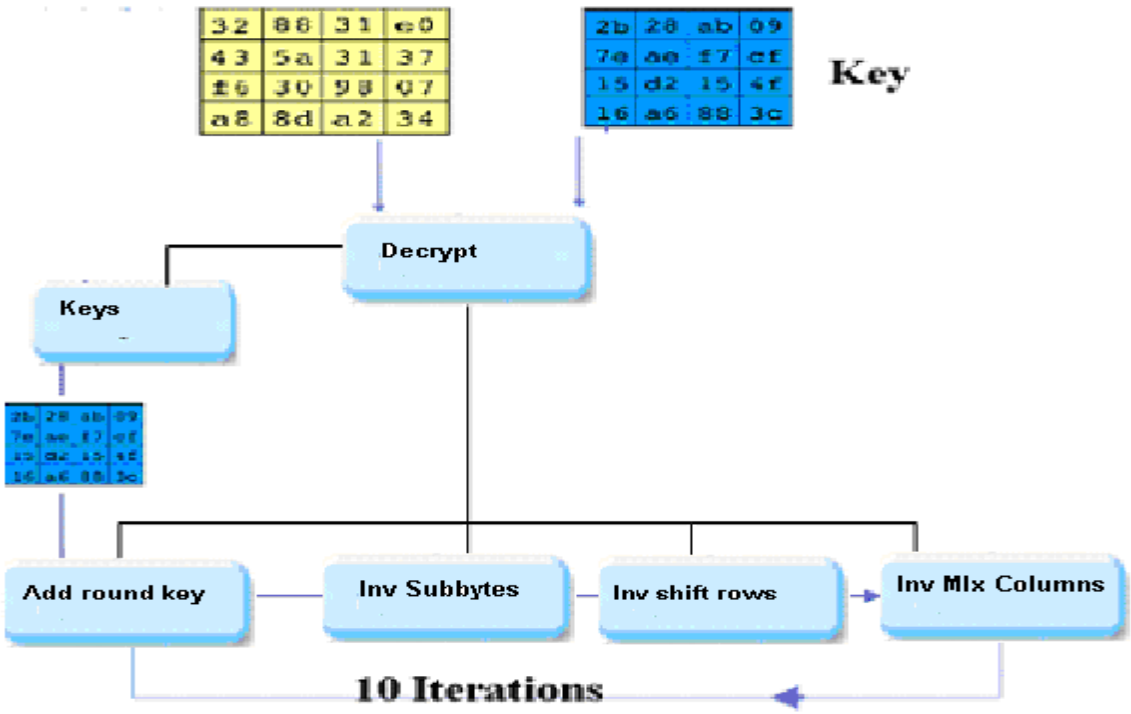


Figure 5.6 Modular Diagram OF Decrypt

5.4.2.1 Inverse Subbytes

The variable Sbox contains the Sbox used in Subbytes. The S-Box is formed by taking the multiplicative inverse in the finite field and then applying the affine transformation in the finite field GF (2^8) the element {00} is mapped to itself. The Matlab function Inverse Subbytes uses the code book method and uses an already stored Sbox hence increasing the execution speed. This function accepts the state (which is formed by breaking the speech in the blocks of 128 bit). The function has an already stored Sbox through which the substitution is done.

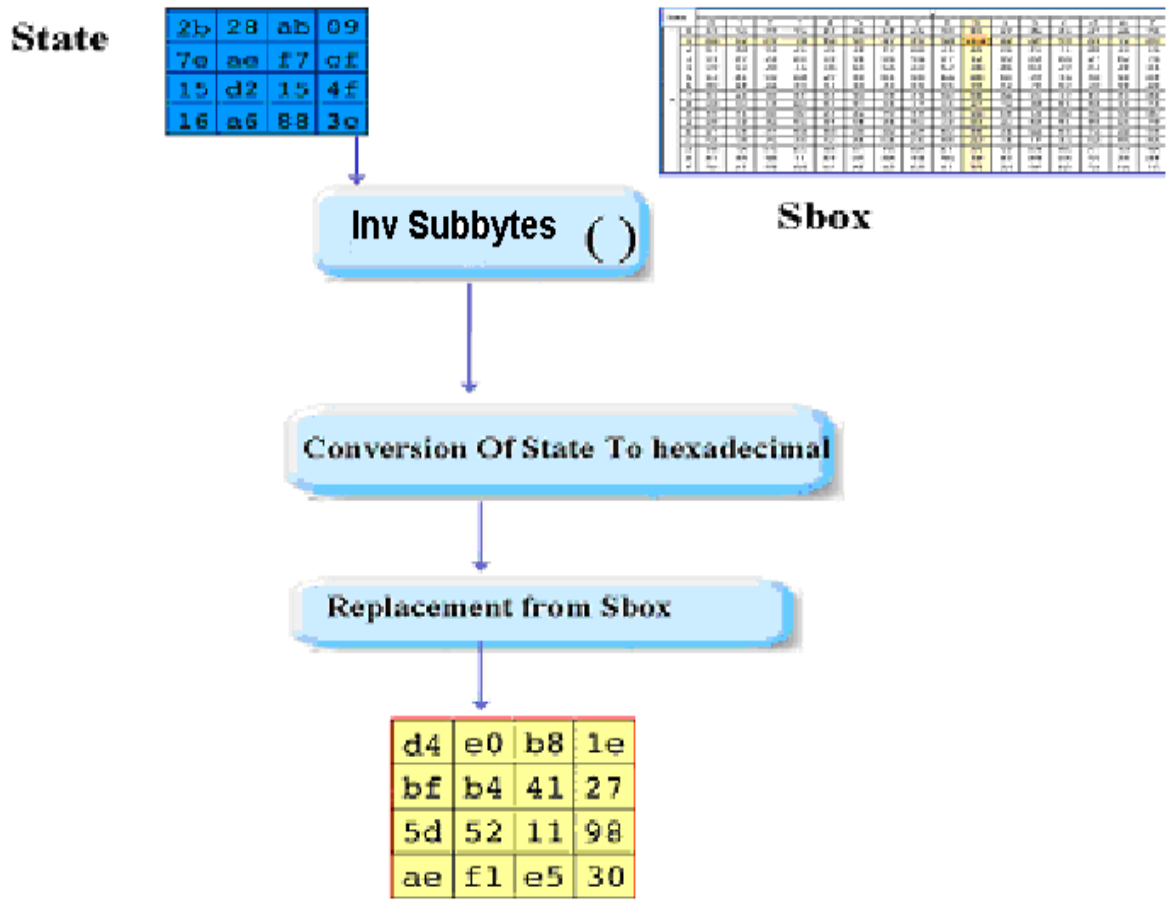


Figure 5.7 Modular description of Inv Subbytes

The Matlab code for Inverse Subbytes is given as under.

```
function state_s=invsubbytes(state);

  [m,n]=size(state);

  z=dec2hex(state);

     for i=1:(m*n)

         state_s(i)=invsbox(hex2dec(z(i,1))+1,hex2dec(z(i,2))+1);
     end
state_s=reshape(state_s,m,n);
 state_sd=dec2hex( state_s);
```

Figure 5.8 Matlab Function invsubbytes

5.4.2.2 Inverse Shiftrows.

In Inverse Shiftrows, the rows of the State are cyclically shifted over different offsets. Row 0 is not shifted; Row 1 is shifted over 1 byte, row 2 over 2 bytes and row 3 over 3 bytes. In Matlab this is achieved by a very simple approach of matrix re assignment. The code for this reassignment is given below.

```
function state_s=invshiftrows(state)

a = state;

state _s=[a(1,1) a(1,2) a(1,3) a(1,4)
        a(2,4) a(2,1) a(2,2) a(2,3)
        a(3,3) a(3,4) a(3,1) a(3,2)
        a(4,2) a(4,3) a(4,4) a(4,1)];
```

Figure 5.9 The Matlab Function Inverse Shift rows.

Graphically the transformation can be shown as

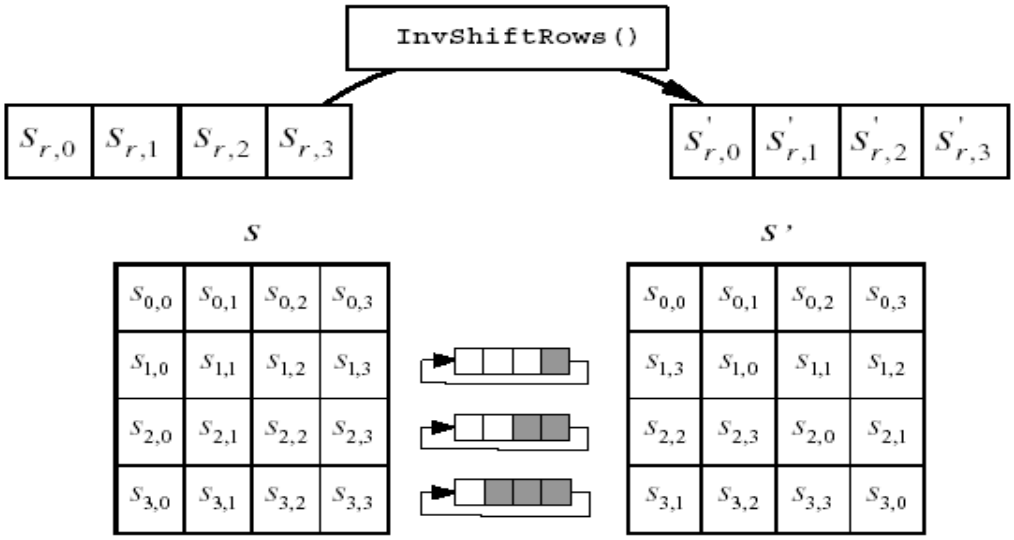


Figure 5.10 Matrix operations In Shift rows.

5.4.2.3 Inverse Mixcolumns.

The function Inverse Mixcolumns multiplies each column of the state with the irreducible polynomial given by a fixed matrix.

The multiplication in the finite field is different from the ordinary multiplication, this is achieved by the function polymult( ) as shown in the figure.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Figure 5.11 Matrix operations In inverse Mixcolumns

The code for the function Inverse Mixcolumns is given as

```
function state_m=invmixcolumns(state)
x= [14   11   13    9
    9   14   11   13
   13    9   14   11
   11   13    9   14];
mod_pol = bin2dec ('100011011');
for i_col_state = 1 : 4
        for i_row_state = 1 : 4
            temp_state = 0;
                for i_inner = 1 : 4
                 temp_prod   =   poly_mult   (x(i_row_state,   i_inner),state(i_inner,
i_col_state),mod_pol);
                 temp_state = bitxor (temp_state, temp_prod);
    end
                 state_m(i_row_state, i_col_state) = temp_state;
   end
   end
```

Figure 5.12 Matlab Code for Inv Mixcolumns

5.4.3 Key Scheduling.

The key scheduling is done with the help of two important functions

- Keys
- Key expansion.

A brief description of the working of these two functions is explained below.

5.4.3.1 Keys

The function keys serves to generate all round key and store them. The function iteratively call the function keyexp() which generates the round keys.

```
function z=keys(k);
for i=1:10
    keys=keyexp(k,i);
    k=keys;
    if i==1
      z=k;
    else
      z=cat(2,z,k);
    end
end
```

Figure 5.13 Matlab Code for Key expansion

5.4.3.2 Key expansion

The function key expansion serves to give the key for each round. The function accepts at its input the round number and the round key of the previous round. In case of round one the input is the cipher key which is being expanded. The function has predefined look up tables for rcon. The function generates the rotword from the fourth column of the key and then calls subbytes for its byte substitution and xors this with the column of the rcon that corresponds with the round number and the first column of the key to generate the first column of the new roundkey. The rest of the three columns of the new round key are generated by simply xor of the i th column with the i-1 column. The round key for each round is generated in a similar fashion. The same is illustrated with the help of a block diagram.

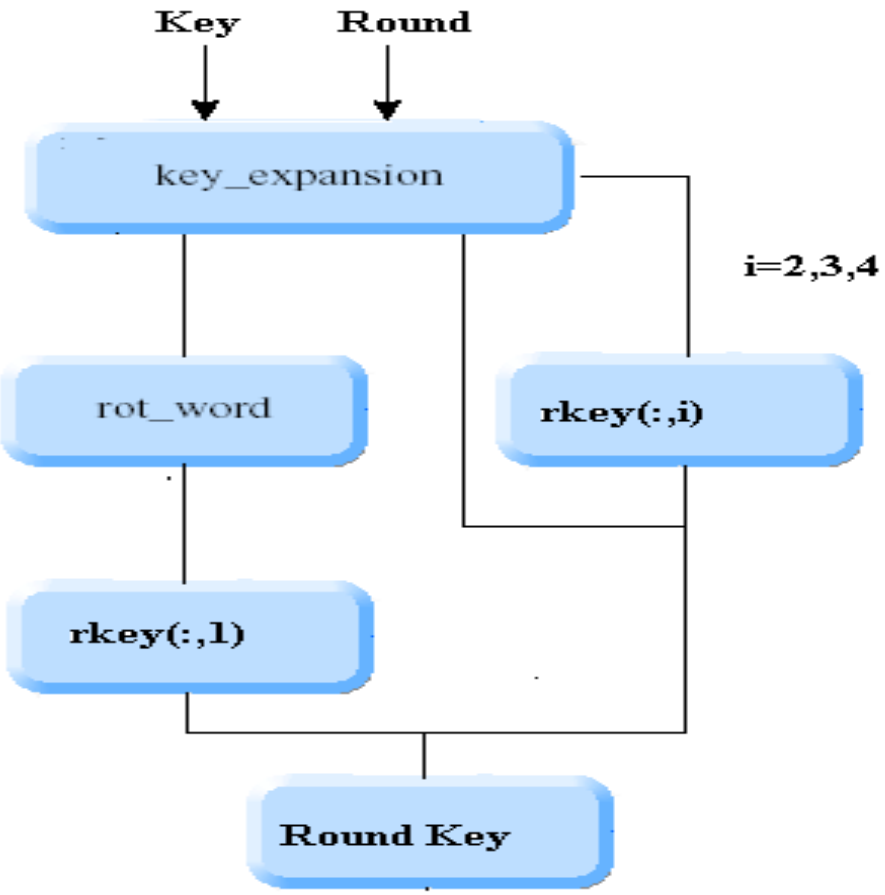


Figure 5.14 Modular Description of Key expansion routine

5.5 The Speech Encryption System.

5.5.1. The Cipher

The Matlab function cipher ( ) serves to input the recorded audio to the function encrypt. It accepts the audio file and the encryption key from the user. It converts the audio to the mu-law encoded state using the Matlab built in function l**in2mu** which convert audio ranging in amplitudes from -1 to 1 to mu-law encoded form ranging in amplitudes from 0 to 255. It catenates the encrypted packets to form the encrypted audio. The iterations depend upon the sampling rate of the audio. As the audio is being sampled at the Nyquist rate of 8000 samples per second hence the iterations are 500 / second.

```
function msg=cipher (s,f);
z=wavread(s,8000);
y=lin2mu(z);
a=1:16:8000;
b=16:16:8000;
for i=1:500
   z=y(a(1,i):b(1,i),1);
   m=reshape(z,4,4);
   m=encrypt(m,f);
   m=reshape(m,16,1);
   m1(a(1,i):b(1,i),1)=m;
end
   msg=m1;
```

Figure 5.15 The Matlab Code for Cipher.

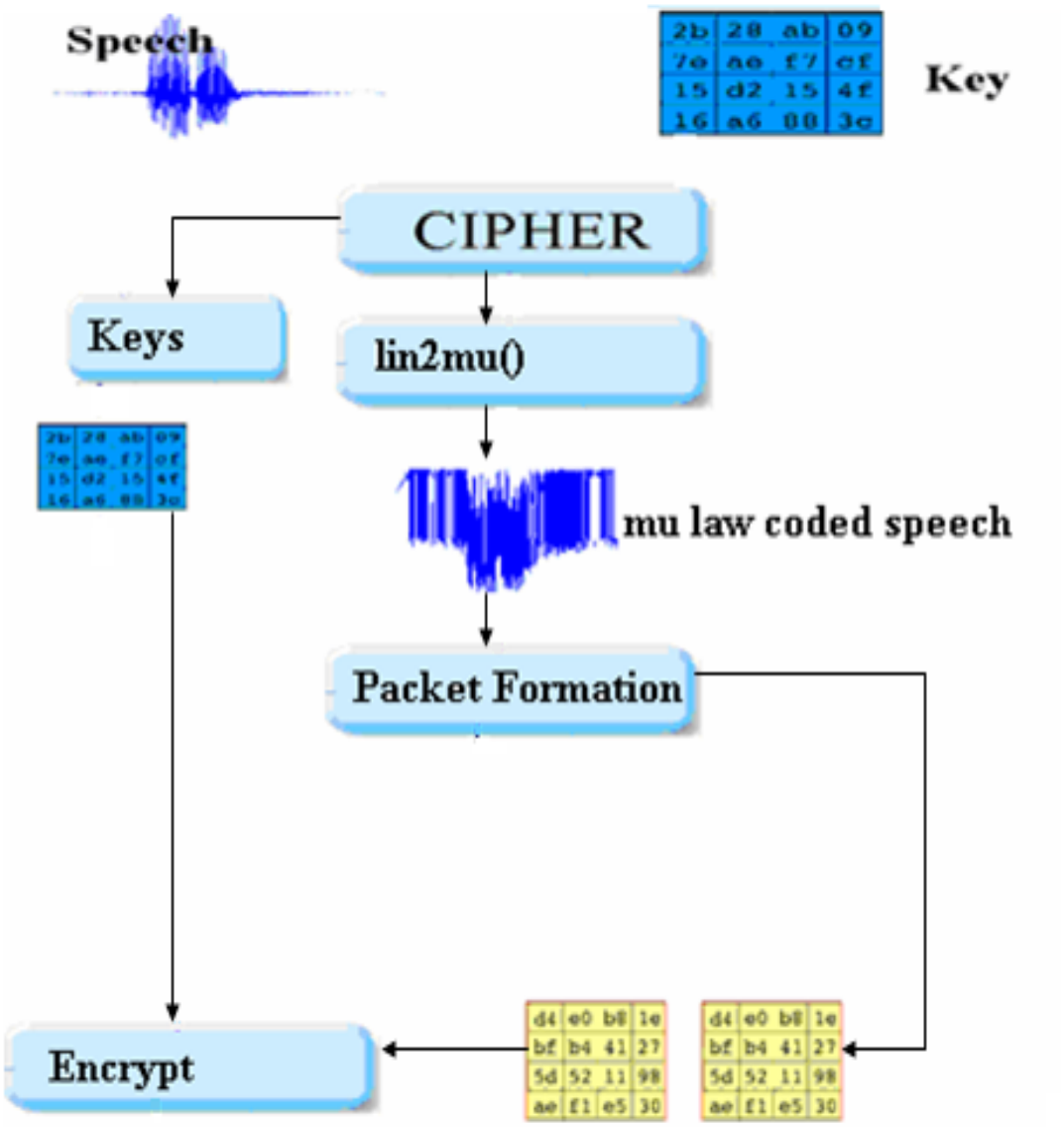


Figure 5.16 Modular Diagram of the speech encryption system

The block diagram clearly shows the functionality of the function cipher( ).The input audio is converted to mu-law coded speech and it is broken into blocks of 128 bit each. These blocks are sequentially passed to the function encrypt which serves to convert them into their respective AES ciphered form. The function cipher() the function encrypt( ) and passes the packets of mu law coded speech to it to be converted into ciphered form.

The functionality of the function encrypt is described in the section 5.4.1.1.

5.5.2 The Decipher

The Matlab function decipher ( ) serves to input the encrypted speech to the function decrypt. It accepts the encrypted audio file and the encryption key from the user. It serves to break the encrypted speech into blocks or packets and passes them to the function decrypt ( ) for decryption. Finally the decrypted audio is converted back to linear format using the Matlab built in function mu2lin(). The output of the decipher function is the original speech.

```
function msg=decipher (s,f);
y=s;
a=1:16:8000;
b=16:16:8000;
for i=1:492
z=y(a(1,i):b(1,i),1);
m=reshape(z,4,4);
m=decrypt(m,f);
m=reshape(m,16,1);
msg(a(1,i):b(1,i),1)=m;
end
msg=mu2lin(msg);
```

Figure 5.17 Matlab Code for decipher

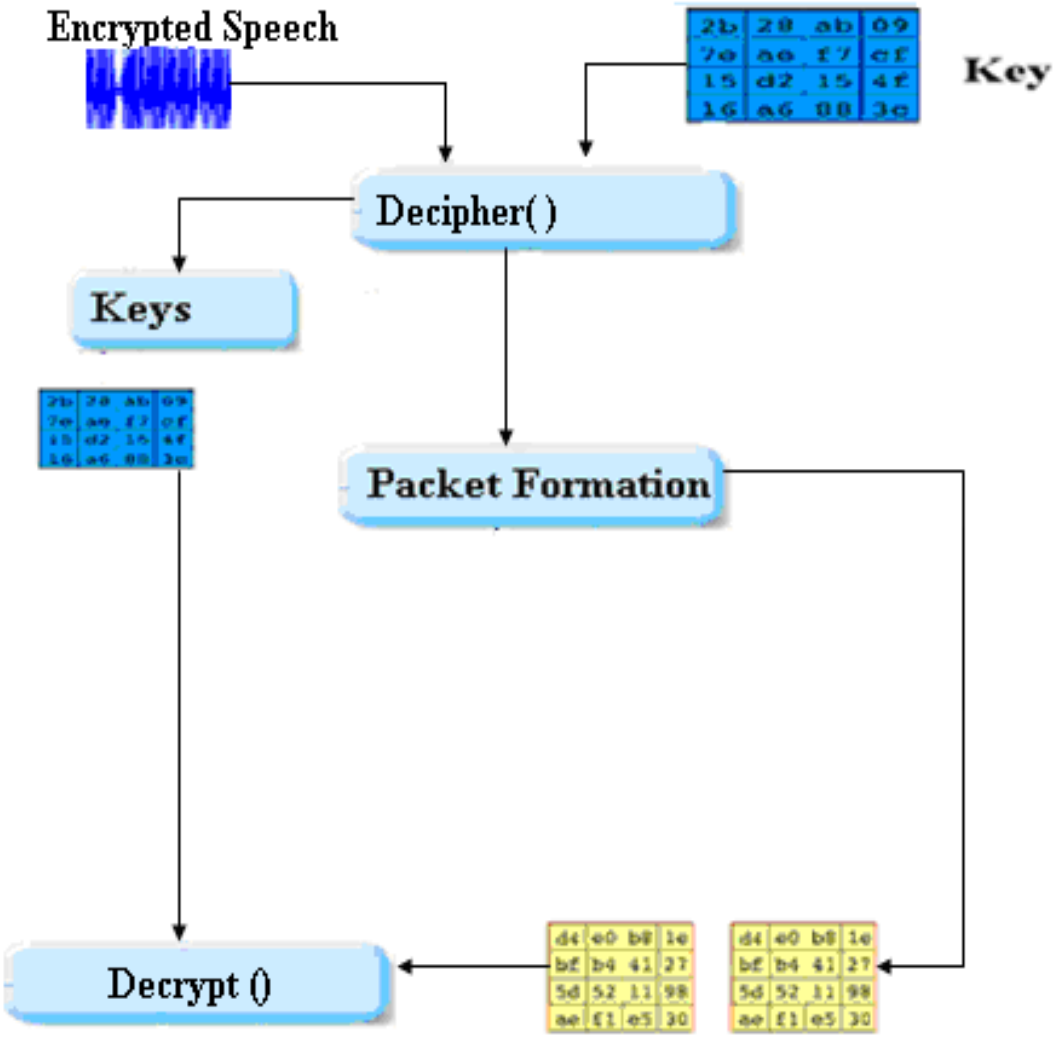


Figure 5.18 Modular Description of the Speech Encryption System.

The block diagram clearly shows the functionality of the function decipher( ).The input audio is converted to mu-law coded speech and it is broken into blocks of 128 bit each. These blocks are sequentially passed to the function encrypt which serves to convert them into their respective AES ciphered form.

5.5.3 The Graphical User Interface.

For the ease of usage and understanding of the users, the AES core designed in MATLAB was put to use in making a graphical user interface. The interface is basically used for speech encryption which is shown in the figure.

The user has the options to enter the name of the speech file and then enter an encryption key. Once the data inputs are complete the user can press the Plain speech tab to see the plot of the plain speech and play the speech in the wav file format.

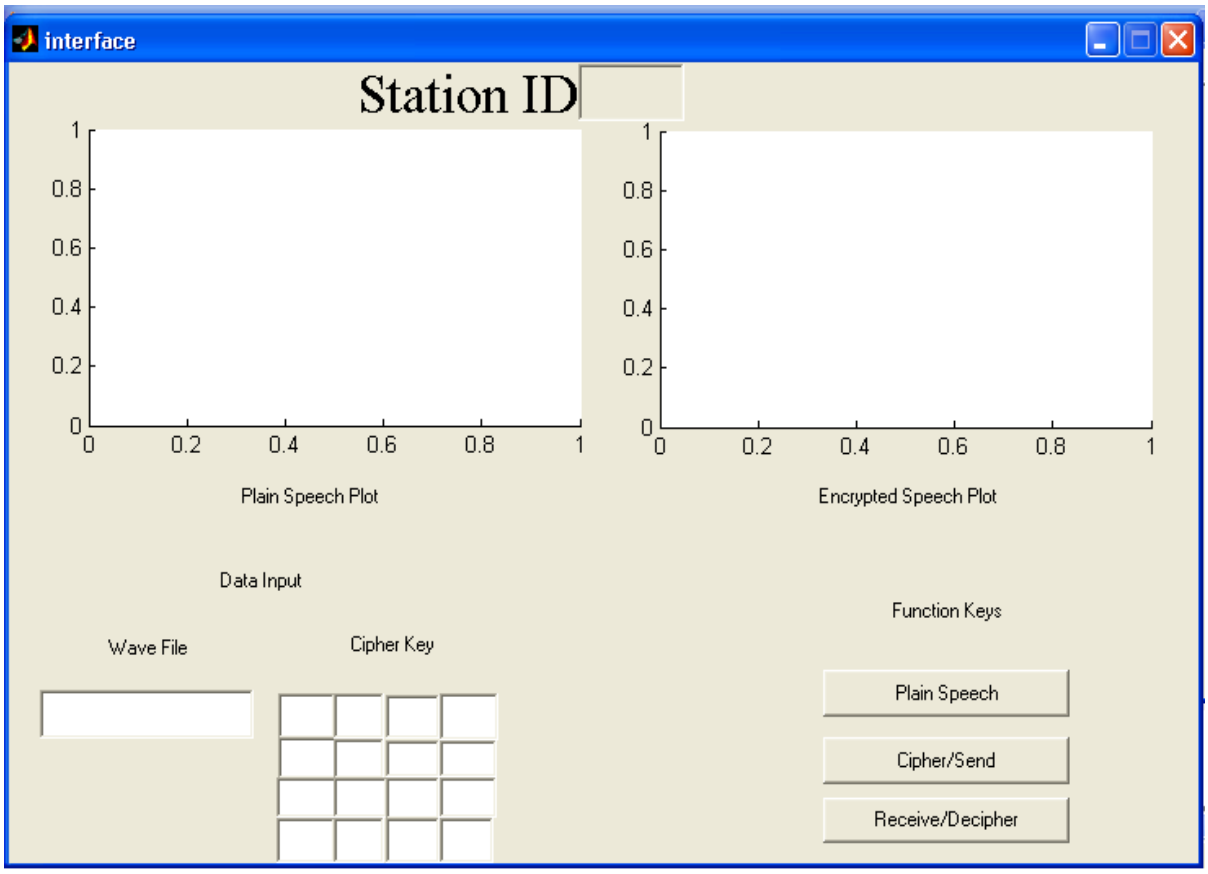


Figure 5.19 Figure MATLAB GUI

The user can then press the Cipher/send button to encrypt the speech and then send it to the remote end. At the remote end the user presses the receive / decipher to receives the encrypted message and decipher it at the remote end. The user at the remote end

has to be in the knowledge of the encryption key in order to decrypt the message correctly. Even a single bit error causes the message to be decrypted incorrectly.

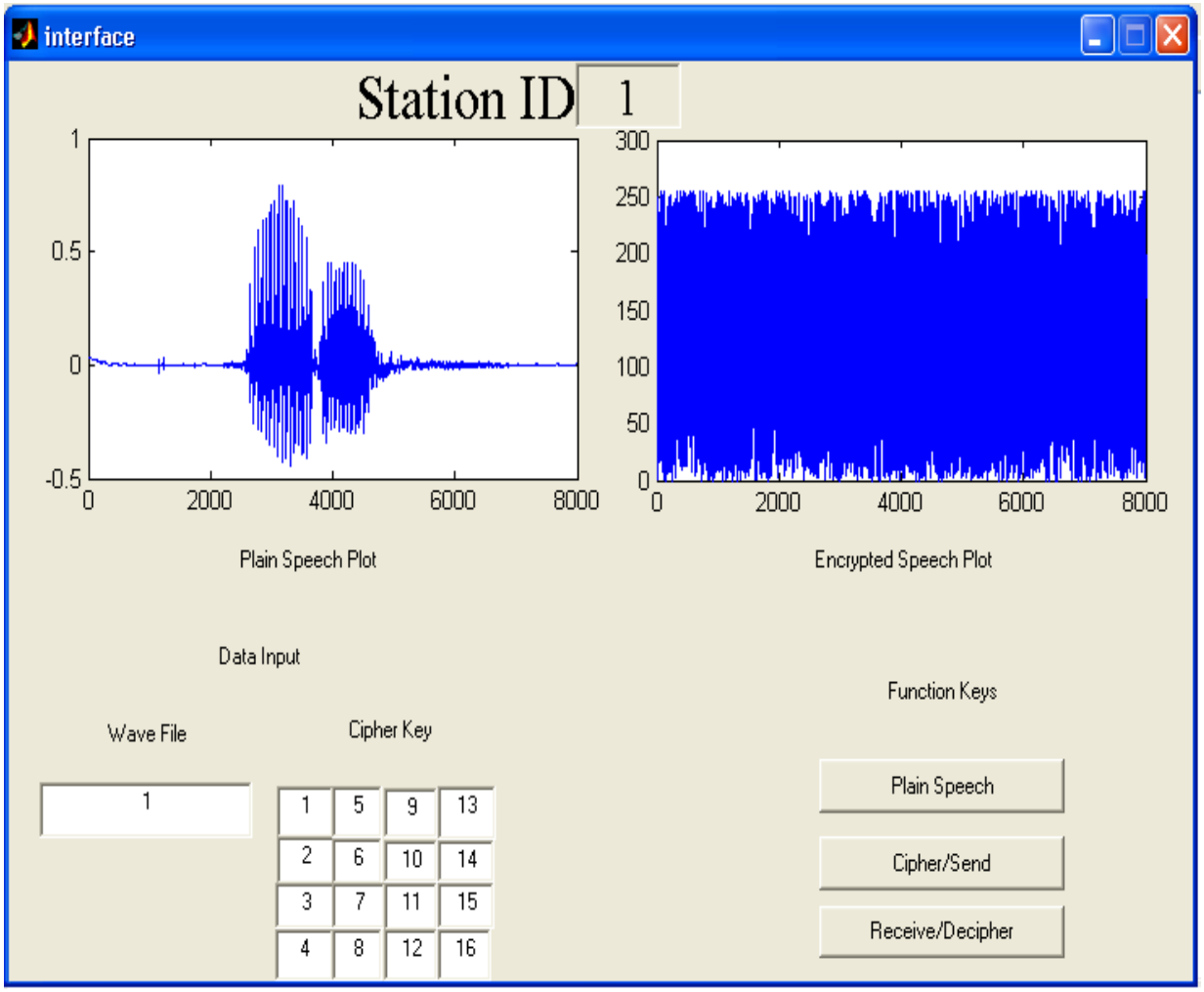


Figure 5.20 Figure MATLAB GUI in operation.

5.6 Feasibility of AES for Image Encryption

With the proliferation of the Internet and maturation of the digital signal processing technology, applications of digital imaging are prevalent and still continuously and rapidly increasing today. Its coverage is literally penetrating all aspects of our daily lives, ranging from digital cinemas, to remote-sense imaging, to medical image processing. And yet the main obstacle in the widespread deployment of digital image services has been enforcing security and ensuring authorized access

to sensitive data. In this regard, a direct solution is to use an encryption algorithm to mask the image data streams, which has led to the celebrated number-theory based encryption algorithms such as DES (Data Encryption Standard), IDEA (International Data Encryption Algorithm) and RSA (developed by Rivest, Shamir and Adleman) . However, these encryption schemes are not ideal for image applications, due to some intrinsic features of images such as bulk data capacity and high redundancy which are troublesome for traditional encryption. Moreover, these encryption schemes require extra operations on compressed image data, thereby demanding long computational time and high computing power. In real-time communications, due to their low encryption and decryption speeds, they may introduce significant latency. Compared with text encryption, which most existing encryption standards aim at, image encryption (or more generally, multimedia encryption) has its own characteristics and special features with many unique specifications. In order to develop effective image encryption techniques, these have to be fully understood.

5.6.1 Particularities of Image Encryption

Unlike text messages, image data have their special features such as bulk capacity, high redundancy, and high correlation among pixels, not to mention that they usually are huge in size, which together make traditional encryption methods diffcult to apply and slow to process. Sometimes, image applications also have their own requirements like realtime processing, fidelity reservation, image format consistence, and data compression for transmission, etc. Simultaneous fulfillment of these requirements, along with high security and high quality demands, have presented great challenges to real-time imaging practice. One example in point is the case where one needs to manage both encryption and compression. In doing so, if he

wants to have an image encrypted after its format is converted, say from a TIFF file to a GIF file, he has to implement encryption before compression. However, for a conventional encrypted image, it has very little compressibility. On the other hand, compression will make a correct and lossless decipher impossible, particularly when a highly secure image encryption scheme is used. This conflict between the compressibility and the security is very difficult, if not impossible, to completely solve. Particularities of image encryption may be summarized as follows:

1. High redundancy and bulk capacity generally make encrypted image data vulnerable to attacks via cryptanalysis. Based on the bulk capacity, the opponent can gain enough cipher text samples (even from one picture) for statistical analysis. Meanwhile, since data in image have high redundancy, adjacent pixels likely have similar grayscale values, or image blocks have similar patterns, which usually embed the image with certain, patterns that results in secret leakage. In the plain-image, adjacent 8-pixel blocks often have similar patterns; therefore, the encrypted image still layouts some intelligible patterns.

2. Image data have strong correlations among adjacent pixels, which makes fast data-shuffling quite difficult. Statistical analysis on large amounts of images shows that averagely adjacent 8 to 16 pixels are correlative in horizontal, vertical, and also diagonal directions for both natural and computer-graphical images. According to Shannon's information theory, secure cryptosystem should fulfill a condition on the information entropy, $E(P|C) = E(P)$, where P stands for plain message and C for ciphered message, that is, the ciphered (i.e., encrypted) image should not provide any information about the plain-image. To meet this requirement, therefore, the ciphered image should be presented as randomly as possible. Since a uniformly distributed message source has a maximum uncertainty, an ideal cipher-image should have an

equilibrium histogram, and any two adjacent pixels should be uncorrelated statistically. This goal is not easy to achieve under only a few rounds of permutation and diffusion.

3. Bulk capacity of image data also makes real-time encryption difficult. Compared with texts, image data capacity is horrendously large. For example, a common 24 bits true color image with 512-pixel height and 512- pixel width will occupy 512 X 512 X 24= 768KBytes in space. Thus, a one-second motion picture will reach up to about 19MBytes. Real-time processing constraints are often required for imaging applications, such as video-conferencing, image surveillance, etc. Vast amount of image data put a great burden on the encoding and decoding processes. Encryption during or after the encoding phase, and decryption during or after the decoding phase, will aggravate the problem. If an encryption algorithm ran very slowly, even with high security, it would have little practical value for real-time imaging applications. That is the reason why current encryption methods such as DES, IDEA and RSA are not the best candidates for this consideration.

4. Image encryption is often to be carried out in combination with data compression. In almost all cases, the data are compressed before they are stored or transmitted, due to the huge amount of image data and their very high redundancy. The main challenge is how to ensure reasonable security while reducing the computational cost without downgrading the compression performance.

5. In image usage, file format conversion is a frequent operation. Image encryption is desirable not to affect such an operation. Thus, directly treating image data as ordinary data for encryption will lead to file format conversion impossible. In this

scenario, contents encryption, where only image data are encrypted leaving file header and control information unencrypted, is preferable.

6. Human vision has high robustness to image degradation and noise. Only encrypting those data bits tied with intelligibility can efficiently accomplish image protection. However, conventional cryptography treats all image data bits equally in importance, thus requires a considerable amount of computational power to encrypt all of them, which has proved unnecessary oftentimes.

7. In terms of security, image data are not as sensitive as text information. Security of image is largely determined by the real situation in an application. Usually, the value of the image information is relatively low, except in some special situations like military and espionage applications or video-conferencing in serious business. A very expensive attack of encrypted median data is generally not worthwhile. In practice, many image applications do not have very strict security requirements. Under certain circumstances, protection of the fidelity of an image object is more important than that of its secrecy. An example in point is electronic signature. As another example, in image database applications, only those users who have paid for the service can access large-size images with high resolution. Adversaries may be able to get some small-size images with low resolution by attacks based on cryptanalysis, but those images have little business values – perhaps much cheaper than the cost they spent on preparing and executing the attacks. In the worst case, possible partial leakage of some secrecy in multimedia, within a certain limitations, is always permitted, while for text information this scenario is largely forbidden because it is then quite easy to predict the entire message based on the obtained information from a partial leakage.

Today, there does not seem to be any image encryption algorithm that can fulfill all the aforementioned specifications and requirements. Chaos-based image encryption,

to be further described below, cannot solve all these problems either. However, it can provide a class of very promising methods that can partially fulfill many of these specs and requirements and demonstrate superiority over the conventional encryption methods, particularly in a good combination of speed, security, and flexibility. As will be seen below, through an elaborative design, either chaotic block cipher or chaotic stream cipher can achieve considerably good overall performance.

5.6.2 Image Encryption using Advanced Encryption Standard

The image is read in the RGB format and then is converted into gray scale mainly to scale it down to a smaller size. The Matlab built in command Rgb2gray is used to perform this conversion. RGB2GRAY converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.

Figure 5.21 Original image.

Figure 5.22 Gray scale image.

The image is stored in the uint8 format. Then it is converted to the double format. Afterwards it is broken into Packets of 128 bit each and input to the AES core. The output of the encrypter is as shown in the figure.

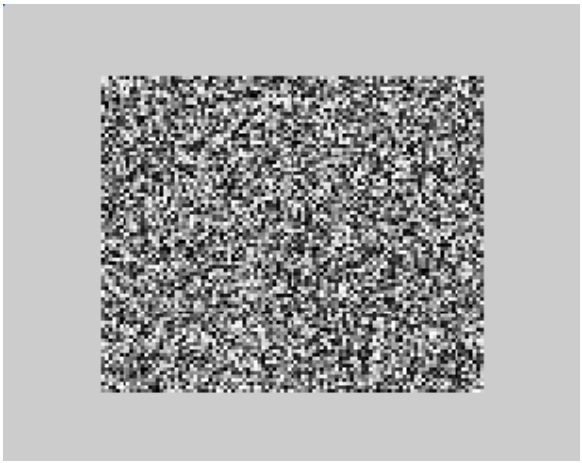


Figure 5.23 Encrypted Image

Figure 5.24 Correctly Decrypted image

At the decrypting end the only the correct knowledge of the key ensures a successful decryption even a one bit error results in a completely obscure result as shown in the figure.

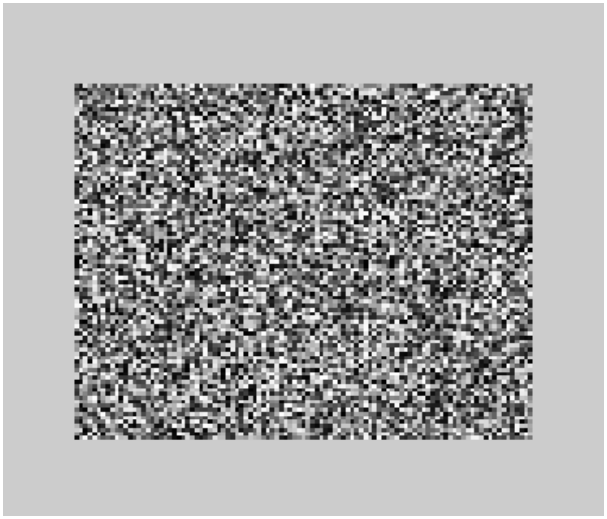


Figure 5.25 Image encrypted with slightly wrong key.

Chapter 6

Implementation of the Advanced Encryption Standard.

6.1 Introduction

The Advanced Encryption Standard is a computationally intensive Algorithm. Hence the real time implementation of the standard is quite a programming task. As defines in the project statement the aim of the project was to implement the advanced encryption standard on the DSK 6711. This was achieved hence the main task of the project was achieved with success. The encrypter and the decrypter were designed and tested on the DSK 6711 and they were found to be fully functional however due to hardware constraints and the unavailability of requisite expertise a real time secure communication could not be achieved between two DSK 6711.

However to prove the practicality of the idea a peer to peer communication model was achieved using the Personal Computers communicating through the serial port. The C code was optimized for this purpose. In addition the serial port communication was understood. The result was a real time secure communication between two computers which was in accordance with the specifications of the advanced encryption standard.

Hence the implementation of the advanced encryption standard were two fold.

- Implementation on the DSK 6711
- Implementation in a Peer to Peer Communication Model between PCs via the

serial port.

The following sections explain the working of the DSK 6711. In addition a brief introduction has been provided regarding the programming procedures on the DSK 6711. A brief introduction to the Code Composer Studio is also provided which the Integrated Development environment (IDE) is for the programming of the DSK 6711.

6.2 DSK 6711 and the code composer studio.

6.2.1 The DSP starter kit

The TMS320C611 (‘6711) DSP is a 32-bit, 150 MHz, floating point digital signal processor. It is based on the very long instruction word (VLIW) architecture. Each VLIW is composed of eight 32-bit instructions, meant for processing by eight pipelined processing units, in parallel. The CPU delivers 1200 MFLOPS (million floating-point operations per second) and 600 MIPS (million instructions per second) of processing power. On-chip memory consists of 32 Kbytes of program memory, 32 Kbytes of data memory, and 8 Kbytes of cache memory. The ‘6711 features two 32-bit timers, and two multi-channel buffered serial ports (MCBSP), which can be used for high-speed communication with external devices. This and other high-performance floating-point processors are extensively deployed in cellular base station transceivers. The DSP starter kit (DSK) for the ‘6711 is a low cost test-bed for TI’s high-end floating point DSPs. As shown in Figure 3.6, this DSK has a parallel-port PC interface, two 80-pin input-output (IO) digital connectors for interfacing with other external devices, and JTAG (joint test action group) connectors for testing. The board also includes 128 Kbytes of flash ROM, and 16 Mbytes of RAM. The ‘C6711

DSK is capable of transferring up to 35 Mb/s of data through each of its MCBSP ports.

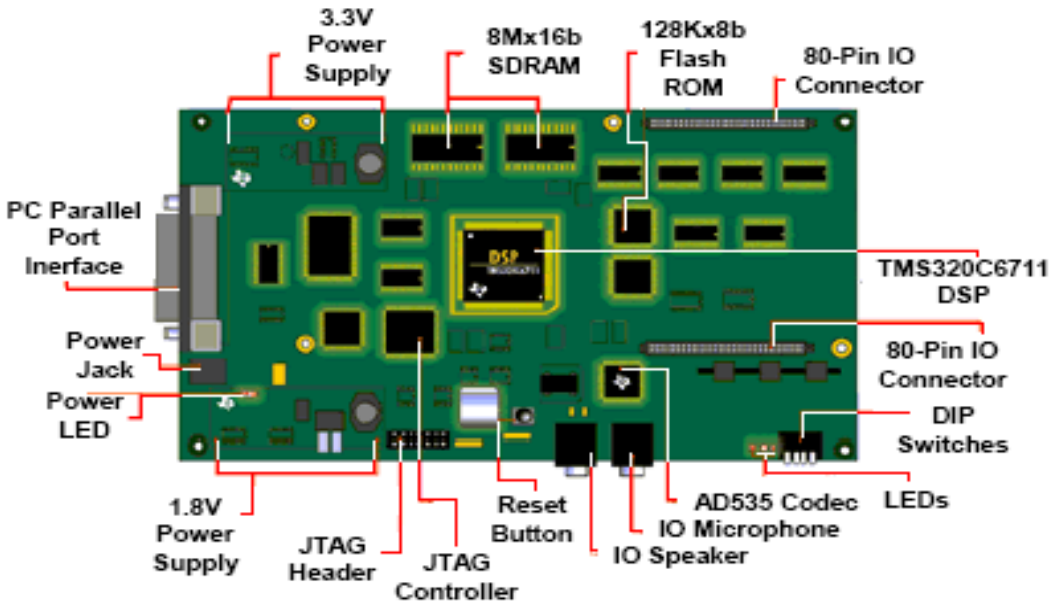


Figure 6.1 DSK 6711

6.2.2 Code Composer Studio.

The Code Composer Studio (CCS) software which is an integrated development environment (IDE) for editing programs, compiling, linking, download to target (i.e., to the DSK board) and debugging. The CCS also includes the DSP/BIOS real-time operating system. The DSP/BIOS co de considerably simplifies the code development for real-time applications which include interrupt driven and time scheduled tasks. Code Composer Studio (CCS) is an integrated development environment customized for the TI family of DSPs. It includes an editor, debugger, project manager, compiler, assembler, linker, and simulator for code generation and testing. What sets it apart from other compiler software is its DSP/BIOS real-time operating system.

DSP/BIOS provide a graphical interface for static system setup, such as initialization of system peripherals. It also allows real time data exchange (RTDX) with the host computer. Interrupts and other tasks can be scheduled through the DSP/BIOS

interface. Furthermore, it allows the real-time monitoring and analysis of system parameters through graphical tools such as spectrum and time-domain plots of variables, or the timing diagrams.

The TMS320C6000 family of pipelined DSPs' complex architecture limits the practicality of assembly coding. Thus most programming is done in high-level languages such as C and C++. The CCS C compiler is touted to be 80% as efficient as hand-coded assembly. The popular signal processing software package, MATLAB, now includes a TI DSP toolbox. This allows the compilation and interfacing of MATLAB code directly to the DSP, by bypassing CCS.

One of the advantages of using TI DSPs is their popularity. There is a treasure trove of programming resources and examples for download on the Internet. The TI website features optimized function libraries free for download. Furthermore, there are archived newsgroups dedicated to each family of the TI line of DSPs, where users from across the world discuss DSP related problems and anecdotes.

6.2.3 Usage overview

Working with the DSK involves the following steps:

• The first step is the algorithm development and programming in C/C++ or assembler. We will only consider coding in C in this document. The program is typed using the editor capabilities of the CCS. Also the DSP/BIOS configuration is performed in a special configuration window. • When the co de is finished it is time to compile and link the different code parts together. This is all done automatically in CCS after pressing the Build button. If the compilation and linking succeeded the finished program is downloaded to the DSK board.

• The operation of DSK board can be controlled using CCS, i.e., CCS also works as a debugger. The program can be started and stopped and single-step. Variables in the

program can be inspected with the Watch functionality. Breakpoints can be inserted in the code.

6.2.4 Getting started with the 6711DSK and CCS

In this section you will be introduced to the programming environment and will download

and execute a simple program on the DSK.

6.2.4.1 Connecting the PC and DSK

The DSK is communicating with the host PC using the parallel-port interface.

- Connect the parallel port of the PC to the parallel port connector on the DSP.
- Power up the DSK by inserting the AC power cable to the black AC/DC converter.

Start Code Composer Studio by clicking on the CCS-SDK2 (C'6000) icon on the workspace.

6.2.4.2 Getting familiar with CCS

CCS arranges its operation around a project and the project view window is displayed to the left. Here all files which related to a certain project are present. At the bottom of the CCS window is the status output view in which status information will be displayed such as compile or link errors as well as other types of information. The window has several tabs for different types of output.

6.2.4.3 Running the code

After a successful build command (compile, link and download) the program is now resident in the DSK and ready for execution. A disassembly window appears showing the Assembly language instructions produced by the compiler.

At the bottom CPU HALTED message appears indicating that the CPU is not executing any code. A few options are available depending on if you want to debug or just run the code. To simply run the code do the following steps.:

• Select Debug.Run to start the execution. At the bottom left you will see the status CPU RUNNING indicating that the program is executed.

• To stop the DSP select Debug. Halt.

• If you want the restart the program do Debug. Restart followed by Debug. Run

6.2.4.4 Use of printf

The printf command which prints information from the DSP to the Stdout window in CCS is not real-time capable. Hence it cannot be used when the DSP is operating in real-time and have hard deadlines to meet. The important tool which was used to view the variables during execution during real time was the watch window utility.

6.3 Implementation of the Advanced Encryption Standard.

This section discusses the real time implementation of the advanced encryption standard on the two platforms

- TI DSK 6711
- Personal Computers.

6.3.1 Implementation on DSK 6711.

For the real time implementation on TI DSK 6711 following major steps were done.

- Code Optimization for TI DSK 6711.
- Implementation of the encrypter and decrypter.
- Communication between Two DSK 6711.

6.3.1.1 Code Optimization, Transformation from C++ to C.

The programming effort done initially was without the knowledge of the functionalities of the DSK 6711 and the CCS. However C++ code for the encrypter and the decrypter was written. They both were dealt with as separate modules however later on they were merged as a single functionality. A function based approach was used in the programming. Each step was programmed as a single function. Generally the global variables were used and the arguments were not passed to the functions. Once the code was transferred to the DSK 6711 it was found that the CCS did not support the <iostream.h>. Hence the code was transformed from C++ to C. The main change was the use of the inline functions. Also the CCS did not allow variable declarations at certain places in the function definition. The code was virtually re-written in the debugging effort. Also the DSK 6711 did not allow the luxury of using as many variables as one wishes so the number of variables was reduced drastically to improve efficiency. The ma\in optimization effort could not be realized as the profile based compilation feature of the CCS was not working. This feature enables the programmer to identify the time consuming portions of the code. Hence the optimization level that was achieved could not be measured statistically.

6.3.1.2 Implementation of the encrypter and decrypter.

The encrypter and the decrypter were successfully implemented on the DSK 6711. They were made to function as two separate modules, however the integration of the two is also possible and was realized later on. The test vectors given in the FIPS 197 were verified and both the modules were found to be working and yielding the required results.

6.3.1.3 Audio Communication between Two DSK 6711.

The audio communication between two DSK 6711 was achieved. This requires the following steps.

- Audio Capture.
- Play back
- Transmission and Reception.

The following block diagram illustrates the functionality of the audio capture and output to the speaker.

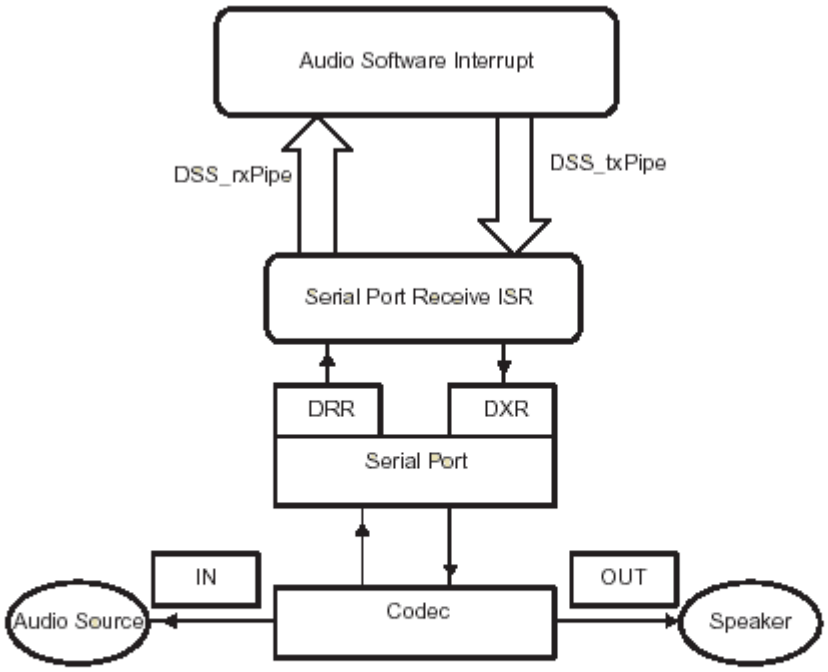


Figure 6.2 Block Diagram Of audio capture

Two pipe objects are used to exchange data between the software interrupt and the serial port connected to the codec.

These pipes are DSS_rxPipe and DSS_txPipe. Data input from the codec flows from the serial port Interrupt Service Routine (ISR) through DSS_rxPipe to the software

interrupt, where it is copied to DSS_txPipe Then it is sent back to the serial port ISR to be transmitted out through the codec, the

ISR for the serial port receive interrupt copies each new 32 bit data sample in the

Data Receive Register (DRR) to a frame from the DSS_rxPipe pipe object. When the frame is full, the ISR puts the frame back into DSS_rxPipe to be read by the audio function.The output from the speaker of the transmitting DSK 6711 is given to the input of the receiving DSK 6711. A similar audio capture utility is functioning at the remote end that enables the communication.The processor of the DSK 6711 is a 150 MHZ chip which was found to be insufficient for our use.

When the two modules described in sections 6.3.1.3 and 6.3.1.2 were made to integrate the system was unable to give the required results. Hence to realize a real time implementation the shift towards the use of PCs as a platform was inevitable.

6.3.2 Implementation on Personal computers.

For the real time communication on the personal computers and the secure communication following steps were performed.

- Serial Port Communication.
- Real time encrypted data transmission and reception.

6.3.2.1 Serial Port Communication

Designing an interface between systems is not a simple or straight-forward task. Parameters that must be taken into account include: data rate, data format, cable length, mode of transmission, termination, bus common mode range, connector type, and system configuration. Noting the number of parameters illustrates how complex this task actually is. Additionally, the

interface’s compatibility with systems from other manufacturers is also critically important. Thus, the need for standardized interfaces becomes evident. Interface Standards resolve both the compatibility issue, and ease the design through the use of non-custom standardized Drivers and Receivers.

Standards have been developed to insure compatibility between units provided by different manufacturers, and to allow for reasonable success in transferring data over specified distances and/or data rates. The Electronics Industry Association (EIA) has produced standards for RS485, RS422, RS232, and RS423 that deal with data communications. Suggestions are often made to deal with practical problems that might be encountered in a typical network. EIA standards where previously marked with the prefix "RS" to indicate recommended standard; however, the standards are now generally indicated as "EIA" standards to identify the standards organization.

6.3.2.2 RS–232 Specifications

RS–232 is a “complete” standard. This means that the standard sets out to ensure compatibility between the host and peripheral systems by specifying 1) common voltage and signal levels, 2)common pin wiring configurations, and 3) a minimal amount of control information between the host and peripheral systems. Unlike many standards which simply specify the electrical characteristics of a given interface, RS–232 specifies electrical, functional, and mechanical characteristics in order to meet the above three criteria

6.3.2.2.1 Electrical Characteristics.

The electrical characteristics section of the RS–232 standard includes specifications on voltage levels, rate of change of signal levels, and line impedance. The original RS–232 standard was defined in 1962. As this was before the days of TTL logic, it should not be surprising that the standard does not use 5 volt and ground logic levels. Instead, a high level for the driver output is defined as being +5 to +15 volts and a low level for the driver output is defined as being between –5 and –15 volts. The receiver logic levels were defined to provide a 2 volt noise margin. As such, a high level for the receiver is defined as +3 to +15 volts and a low level is –3 to –15 volts. Figure 1 illustrates the logic levels defined by the RS–232 standard. It is necessary to note that, for RS–232 communication, a low level (–3 to –15 volts) is defined as a logic 1 and is historically referred to as “marking”. Likewise a high level (+3 to +15 volts) is defined as logic 0 and is referred to as “spacing”. The RS–232 standard also limits the maximum slew rate at the driver output. This limitation was included to help reduce the likelihood of cross–talk between adjacent signals. The slower the rise and fall time, the smaller the chance of cross talk. With this in mind, the maximum slew rate allowed is 30 V/μs. Additionally, a maximum data rate of 20k bits/second has been defined by the standard.

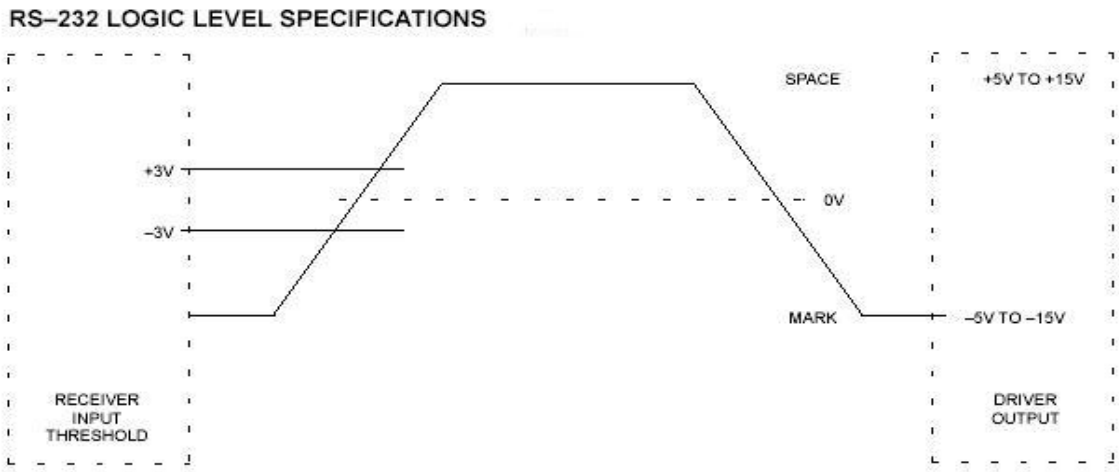


Figure 6.3 Electrical Specifications

Again with the purpose of reducing the chance of cross talk, the impedance of the interface between the driver and receiver has also been defined. The load seen by the driver is specified to be 3k to 7k.. For the original RS–232 standard, the cable between the driver and the receiver was also specified to be a maximum of 15 meters in length. This part of the standard was changed in revision “D” (EIA/TIA–232–D). Instead of specifying the maximum length of cable, a maximum capacitive load of 2500 pF was specified which is clearly a more adequate specification.

6.3.2.2.2 Functional Characteristics.

Since RS–232 is a “complete” standard, it includes more than just specifications on electrical characteristics. The second aspect of operation that is covered by the standard concerns the functional characteristics of the interface. This essentially means that RS–232 has defined the function of the different signals that are used in

the interface. These signals are divided into four different categories: common, data, control, and timing. Table 1 illustrates the signals that are defined by the RS–232 standard. As can be seen from the table there is an overwhelming number of signals defined by the standard. The standard provides an abundance of control signals and supports a primary and secondary communications channel. Fortunately few applications, if any, require all of these defined signals. For example, only eight signals are used for a typical modem. Some simple applications may require only four signals (two for data and two for handshaking) while others may require only data signals with no handshaking.

RS–232 DEFINED SIGNALS

CIRCUIT MNEMONIC	CIRCUIT NAME*	CIRCUIT DIRECTION	CIRCUIT TYPE
AB	Signal Common	–	Common
BA	Transmitted Data (TD)	To DCE	Data
BB	Received Data (RD)	From DCE	
CA	Request to Send (RTS)	To DCE	
CB	Clear to Send (CTS)	From DCE	
CC	DCE Ready (DSR)	From DCE	
CD	DTE Ready (DTR)	To DCE	
CE	Ring Indicator (RI)	From DCE	
CF	Received Line Signal Detector** (DCD)	From DCE	Control
CG	Signal Quality Detector	From DCE	
CH	Data Signal Rate Detector from DTE	To DCE	
CI	Data Signal Rate Detector from DCE	From DCE	
CJ	Ready for Receiving	To DCE	
RL	Remote Loopback	To DCE	
LL	Local Loopback	To DCE	
TM	Test Mode	From DCE	
DA	Transmitter Signal Element Timing from DTE	To DCE	
DB	Transmitter Signal Element Timing from DCE	From DCE	Timing
DD	Receiver Signal Element Timing From DCE	From DCE	
SBA	Secondary Transmitted Data	To DCE	Data
SBB	Secondary Received Data	From DCE	
SCA	Secondary Request to Send	To DCE	
SCB	Secondary Clear to Send	From DCE	Control
SCF	Secondary Received Line Signal Detector	From DCE	

*Signals with abbreviations in parentheses are the eight most commonly used signals.
**This signal is more commonly referred to as Data Carrier Detect (DCD).

Figure 6.4 RS 232 Defined Signals

6.3.2.2.3 MECHANICAL CHARACTERISTICS

The third area covered by RS–232 concerns the mechanical interface. In particular, RS–232 specifies a 25–pin connector. This is the minimum connector size that can accommodate all of the signals defined in the functional portion of the standard. The pin assignment for this connector is shown in Figure. The connector for DCE equipment is male for the connector housing and female for the connection pins. Likewise, the DTE connector is a female housing with male connection pins. Although RS–232 specifies a 25–position connector, it should be noted that often this connector is not used. This is due to the fact that most applications do not require all of the defined signals and therefore a25–pin connector is larger than necessary. This being the case, it is very common for other connector types to be used. Perhaps the most popular is the 9–position DB9S connector which is also illustrated in Figure.

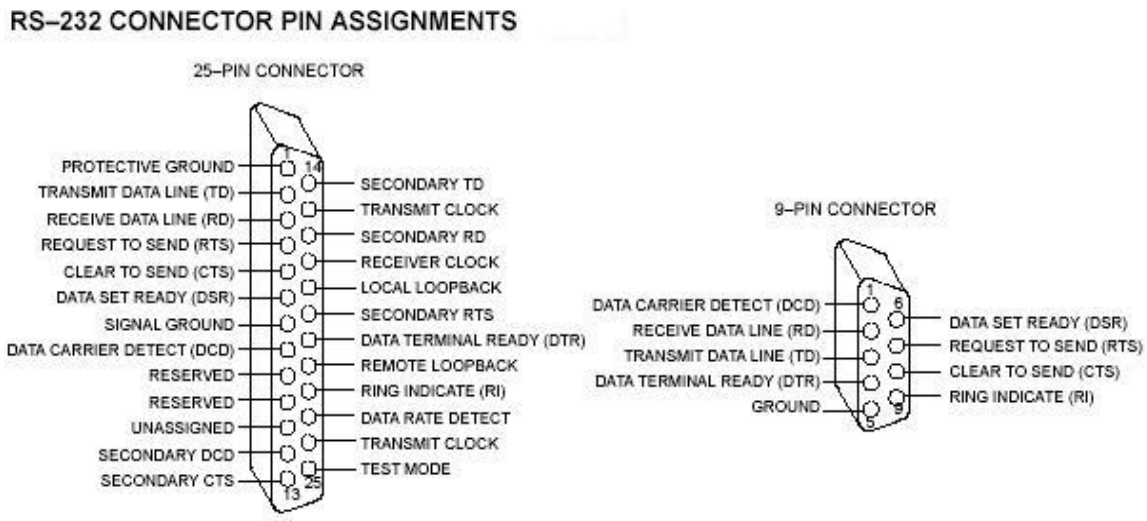


Figure 6.5 RS232 Mechanical Specifications

6.3.2.3 Real time encrypted data transmission and reception.

The AES core programmed in C was now put to use for the encrypted data transmission via the serial port. As imperative AES only accepts data in the block format of 128 bit size. Hence the packetization of the data input had to be done. The inputs to the program are the 128 bit key and the data in the stream format. The following flow diagram explains the flow of the operations.

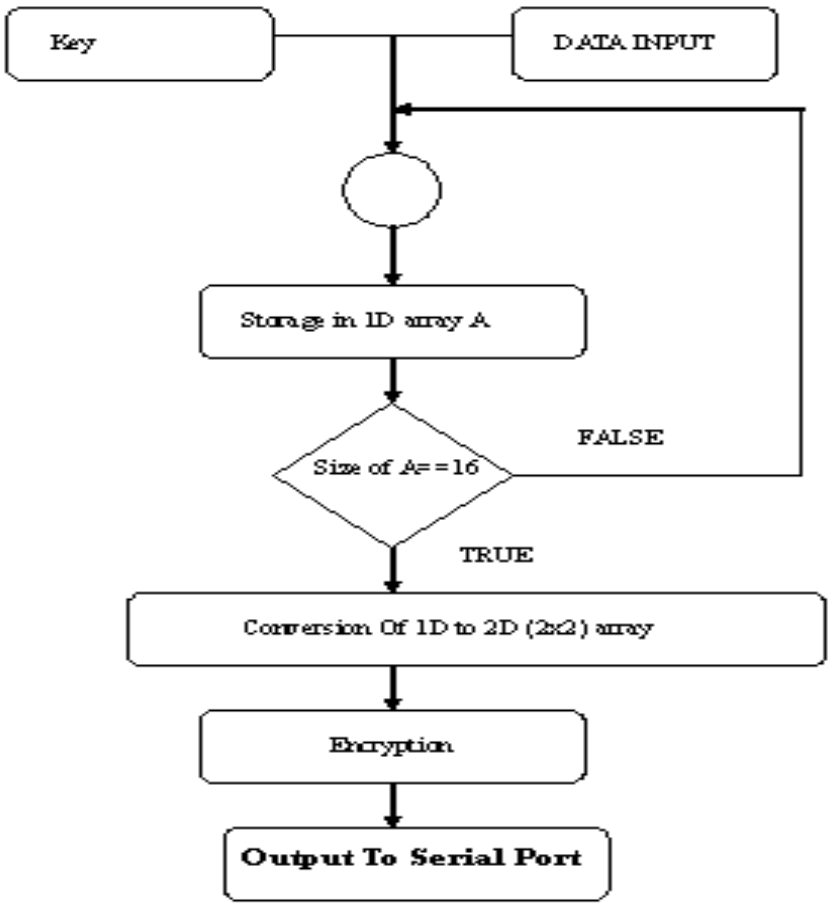


Figure 6.6 Flow of Operations

The key is sent to the key expansion routines and the round keys are generated. These round keys are used in the encryption process. Since the encryption key is entered as a 128 bit packet no additional processing is required. However this is not true in case of the data. The data entered through the keyboard is read and stored in a one dimensional array. The size of the array is checked and one it is 16 values or 128 bits it is reshaped into a 4x4 array into the packets of 128 bit each. This packet of size of 128 bit is sent into the AES core. The AES core encrypts the packet and sends it via the serial port.

Chapter 7

Summary Recommendations and Conclusion

7.1 Summary

The advanced encryption standard forms a very solid base for a secure communication system. The standard has been tested for various types of inputs like text, image and speech and has been found to be very effective. The end result of the project can be summarized as

- The DSK 6711 was programmed to work as an encryption and the decryption device which was found to be working in prefect conditions. The test vectors given in FIPS 197 were verified.
- The DSK 6711 could not be interfaced with a transceiver due to the unavailability of the daughter cards of the DSK 6711.
- Speech encryption scheme was design and simulated in Matlab and was found to give excellent result.
- The encryption and decryption were tested in a real time system using the windows 98 based computers communicating via the serial port.
- A study was undertaken to ascertain the feasibility of AES for image encryption and the results were very encouraging.
- A speech encryption system in C has nearly been implemented. A few logical errors are holding up the end result.

7.2 Recommendations.

For the application and improvement of the project following recommendations are given

- DSK 6711 has a 150 MHZ processor which was found to be in sufficient for the speech encryption in real time. DSP chips with highly improved processing speeds are available now a days which are more suitable for computationally intensive algorithms like AES.
- Daughter cards for the DSK 6711 are readily available and can be procured from Texas Instruments. These cards have better sampling rate and are necessary for applications like images.
- Transceivers should be designed which should be interfaced with the DSK 6711.
- By interfacing an FSK modem with the serial port of the computer the encrypted data can be transmitted over the HF and VHF links. Such a project has been realized while using the DES encryption scheme.
- The AES core implemented can be implemented for network security applications.
- Profile based compilation tool is not available in the DSP lab which is needed to optimize the code.

7.3 Conclusion

Secure communications systems are of great importance these days. Communication security is especially of greater interest for military purposes. Whatever measures be adopted to provide communication security, they always face the looming danger of being breached by hostile sources. Hence the need to implement a fool proof security measure remains a difficult task. The Project undertaken is an endeavor in the same direction. The aim of the project is to develop a secure communication system over wireless and wire line media with optimum guaranteed security. The Advanced encryption standard promises to be the state of the art technology for the next many decades. The option to increase the block size to 192 and 256 bits in case of a cracking of the 128 bit key provides a back up plan

With the help of the improved processors which are readily available in the market AES promises to be suitable for voice, ISDN and HDTV. A hardware implementation of the AES must be pursued for better results.

References.

[1] The Mathworks: Matlab , The Language of Technical Computing. http://www.mathworks.com/products/matlab, (2001).

[2] The Mathworks: Galois Field Computations. http://www.mathworks.com/access/helpdesk/help/toolbox/comm/tutor3.shtml, Communications Toolbox, (2001).

[3] J. Daemen, L. R. Knudsen, and V. Rijmen: The Galois Field GF(28). http://www.ddj.com/documents/s=936/ddj9710e/9710es1.htm, Dr. Dobb's Journal, (October 1997).

[4] V. Rijmen: The block cipher Rijndael. http://www.esat.kuleuven.ac.be/~rijmen/rijndael/, (2001).

[5] J. Daemen, V. Rijmen: AES proposal: Rijndael. http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndaeldocV2.zip, (1999).

[6] J. Daemen: Annex to AES proposal Rijndael. http://www.esat.kuleuven.ac.be/~rijmen/rijndael/PropCorr.PDF, (1998).

[7] National Institute of Standards and Technology: Specification for the Advanced Encryption
Standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, (2001).

[8] National Institute of Standards and Technology: Data Encryption Standard (DES). http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, (2001).

[9] B. Schneier: Applied Cryptography. Addison-Wesley, (1996).

[10] [SPRU305] EVM Technical Reference

[11] [SPRU273B] TMS320C6x Peripheral Support Library Programmer's Reference

[12] [SPRU190] TMS320C6000 Peripheral Reference Guide

[13] [SPRU189E] TMS320C6000 CPU and Instruction Set Reference Guide