TEXT FILE ENCRYPTION USING FFT TECHNIQUE IN Lab VIEW 8.6

Sudha Rani. K[1], T. C. Sarma[2], K. Satya Prasad[3]

[1]*DEPT of EIE, VNRVJIET, Hyderabad, India,* ***sudhasarah@gmail.com***
[2]*Former Deputy Director, NRSA, Hyderabad, India,* ***sarma_tc@yahoo.com***
[3]*Rector, JNTU Kakinada University, Kakinada, India,* ***prasad_kodati@yahoo.com***

Abstract

Encryption has always been a very important part of military communications. Here we deal with digital transmission technique. Digital transmission is always much more efficient than analog transmission, and it is much easier for digital encryption techniques to achieve a very high degree of security. Of course, this type of technique is still not quite compatible with today's technical environment, i.e. most of the telephone systems are still analog instead of digital; most practical digitizers still require a relatively high bit rate which cannot be transmitted via standard analog telephone channels; and low bit rate speech digitizers still imply relatively high complexity and poor quality. Digital transmission adopts "Scrambling" technique. Scrambling methods are considered as important methods that provide the communication systems a specified degree of security, depending on the used technique to implement the scrambling method. There are many traditional scrambling methods used in single dimension such as time or frequency domain scrambling.

Index Terms: *Encryptor, Decryptor, Fast Fourier transforms (FFT)*

----------***----------

1. INTRODUCTION

Encryption has long been used by militaries and governments to facilitate secret communication. It is now commonly used in protecting information within many kinds of civilian systems. For example, the Computer Security Institute reported that in 2007, 71% of companies surveyed utilized encryption for some of their data in transit, and 53% utilized encryption for some of their data in storage. Encryption can be used to protect data "at rest", such as files on computers and storage devices (e.g. USB flash drives). In recent years there have been numerous reports of confidential data such as customers' personal records being exposed through loss or theft of laptops or backup drives. Encrypting such files at rest helps protect them should physical security measures fail. Digital rights management systems which prevent unauthorized use or reproduction of copyrighted material and protect software against reverse engineering are another somewhat different example of using encryption on data at rest.

Encryption is also used to protect data in transit, for example data being transferred via networks (e.g. the Internet, e-commerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. There have been numerous reports of data in transit being intercepted in recent years. Encrypting data in transit also helps to secure it as it is often difficult to physically secure all access to networks. Encryption, by itself, can protect the confidentiality of messages, but other techniques are still needed to protect the integrity and authenticity of a message; for example, verification of a message authentication code (MAC) or a digital signature. Standards and cryptographic software and hardware to perform encryption are widely available, but successfully using encryption to ensure security may be a challenging problem. A single slip-up in system design or execution can allow successful attacks. Sometimes an adversary can obtain unencrypted information without directly undoing the encryption.

2. FAST FOURIER TRANSFORM (FFT)

In this section we present several methods for computing the DFT efficiently. In view of the importance of the DFT in various digital signal processing applications, such as linear filtering, correlation analysis, and spectrum analysis, its efficient computation is a topic that has received considerable attention by many mathematicians, engineers, and applied scientists. Basically, the computational problem for the DFT is to compute the sequence {X(k)} of N complex-valued numbers given another sequence of data {x(n)} of length N, according to the formula

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad 0 \le k \le N-1$$

$$W_N = e^{-j2\pi/N}$$

In general, the data sequence x(n) is also assumed to be complex valued. Similarly, The IDFT becomes

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) W_N^{-nk}, \qquad 0 \le n \le N-1$$

Since DFT and IDFT involve basically the same type of computations, our discussion of efficient computational algorithms for the DFT applies as well to the efficient computation of the IDFT.

We observe that for each value of k, direct computation of X(k) involves N complex multiplications (4N real multiplications) and N-1 complex additions (4N-2 real additions). Consequently, to compute all N values of the DFT requires N^2 complex multiplications and N^2-N complex additions.

Direct computation of the DFT is basically inefficient primarily because it does not exploit the symmetry and periodicity properties of the phase factor W_N. In particular, these two properties are :

$$\text{Symmetry property}: W_N^{k+N/2} = -W_N^k$$
$$\text{Periodicit y property}: W_N^{k+N} = W_N^k$$

The computationally efficient algorithms described in this section, known collectively as fast Fourier transform (FFT) algorithms, exploit these two basic properties of the phase factor.

2.1Radix-2 FFT Algorithms

Let us consider the computation of the $N = 2^v$ point DFT by the divide-and conquer approach. We split the N-point data sequence into two N/2-point data sequences $f_1(n)$ and $f_2(n)$, corresponding to the even-numbered and odd-numbered samples of x(n), respectively, that is,

$$f_1(n) = x(2n)$$
$$f_2(n) = x(2n+1), \qquad n = 0,1,\ldots,\frac{N}{2}-1$$

Thus $f_1(n)$ and $f_2(n)$ are obtained by decimating x(n) by a factor of 2, and hence the resulting FFT algorithm is called a decimation-in-time algorithm.

Now the N-point DFT can be expressed in terms of the DFT's of the decimated sequences as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0,1,\ldots,N-1$$
$$= \sum_{n \text{ even}} x(n) W_N^{kn} + \sum_{n \text{ odd}} x(n) W_N^{kn}$$
$$= \sum_{m=0}^{(N/2)-1} x(2m) W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_N^{k(2m+1)}$$

But $W_N^2 = W_{N/2}$. With this substitution, the equation can be expressed as

$$X(k) = \sum_{m=0}^{(N/2)-1} f_1(m) W_{N/2}^{km} + W_N^k \sum_{m=0}^{(N/2)-1} f_2(m) W_{N/2}^{km}$$
$$= F_1(k) + W_N^k F_2(k), \qquad k = 0,1,\ldots,N-1$$

Where $F_1(k)$ and $F_2(k)$ are the N/2-point DFTs of the sequences $f_1(m)$ and $f_2(m)$, respectively.

Since $F_1(k)$ and $F_2(k)$ are periodic, with period N/2, we have $F_1(k+N/2) = F_1(k)$ and $F_2(k+N/2) = F_2(k)$. In addition, the factor $W_N^{k+N/2} = -W_N^k$. Hence the equation may be expressed as

$$X(k) = F_1(k) + W_N^k F_2(k), \qquad k = 0,1,\ldots,\frac{N}{2}-1$$
$$X(k+\frac{N}{2}) = F_1(k) - W_N^k F_2(k), \qquad k = 0,1,\ldots,\frac{N}{2}-1$$

We observe that the direct computation of $F_1(k)$ requires $(N/2)^2$ complex multiplications. The same applies to the computation of $F_2(k)$. Furthermore, there are N/2 additional complex multiplications required to compute $W_N^k F_2(k)$. Hence the computation of X(k) requires $2(N/2)^2 + N/2 = N^2/2 + N/2$ complex multiplications. This first step results in a reduction of the number of multiplications from N^2 to $N^2/2 + N/2$, which is about a factor of 2 for N large.

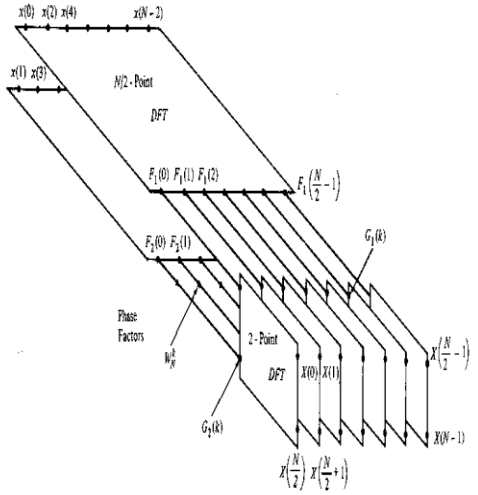

Figure-1.1: First step in the decimation-in-time algorithm.

By computing N/4-point DFTs, we would obtain the N/2-point DFTs $F_1(k)$ and $F_2(k)$ from the relations

$$F_1(k) = F\{f_1(2n)\} + W_{N/2}^k F\{f_1(2n+1)\}, \quad k = 0,1,\ldots,\frac{N}{4}-1; \quad n = 0,1,\ldots,\frac{N}{4}-1$$

$$F_1\left(k+\frac{N}{4}\right) = F\{f_1(2n)\} - W_{N/2}^k F\{f_1(2n+1)\}, \quad k = 0,1,\ldots,\frac{N}{4}-1; \quad n = 0,1,\ldots,\frac{N}{4}-1$$

$$F_2(k) = F\{f_2(2n)\} + W_{N/2}^k F\{f_2(2n+1)\}, \quad k = 0,1,\ldots,\frac{N}{4}-1; \quad n = 0,1,\ldots,\frac{N}{4}-1$$

$$F_2\left(k+\frac{N}{4}\right) = F\{f_2(2n)\} - W_{N/2}^k F\{f_2(2n+1)\}, \quad k = 0,1,\ldots,\frac{N}{4}-1; \quad n = 0,1,\ldots,\frac{N}{4}-1$$

$F(*)$ represents Fourier tr ansform

The decimation of the data sequence can be repeated again and again until the resulting sequences are reduced to one-point sequences. For $N= 2^v$, this decimation can be performed $v = \log_2 N$ times. Thus the total number of complex multiplications is reduced to $(N/2)\log_2 N$. The number of complex additions is $N\log_2 N$.

For illustrative purposes, Figure 1.2 depicts the computation of N = 8 point DFT. We observe that the computation is performed in three stages, beginning with the computations of four two-point DFTs, then two four-point DFTs, and finally, one eight-point DFT. The combination for the smaller DFTs to form the larger DFT is illustrated in Figure 2.3 for N = 8.

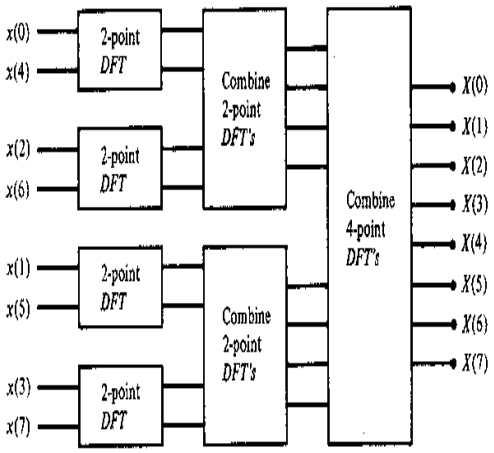

Figure -1.2: Three stages in the computation of an N = 8-point DFT.

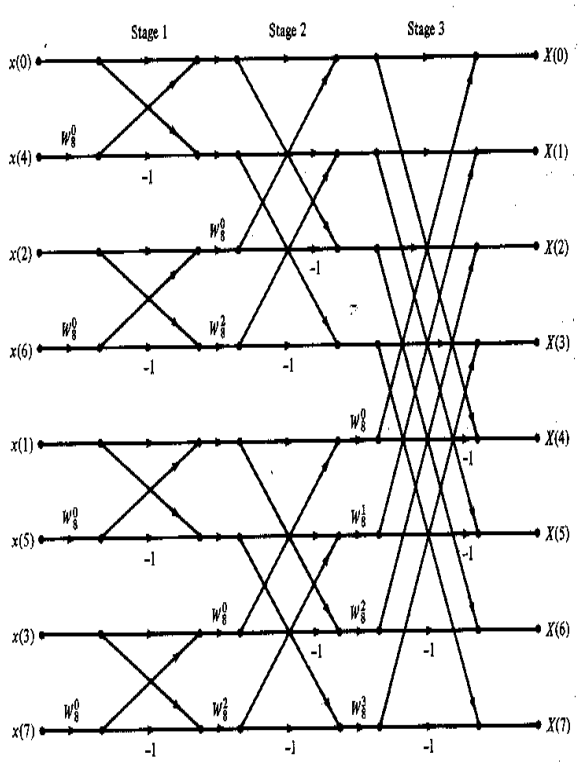

Figure -1.3 Eight-point decimation-in-time FFT algorithms.

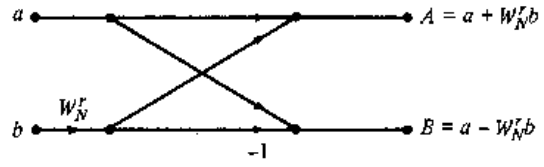

Figure-1.4: basic butterfly computations in the decimation-in-time FFT algorithm.

An important observation is concerned with the order of the input data sequence after it is decimated (v-1) times. For example, if we consider the case where N = 8, we know that the first decimation yields the sequence x(0), x(2), x(4), x(6), x(1), x(3), x(5), x(7), and the second decimation results in the sequence x(0), x(4), x(2), x(6), x(1), x(5), x(3), x(7). This shuffling of the input data sequence has a well-defined order as can be ascertained from observing Figure1.5, which illustrates the decimation of the eight-point sequence.

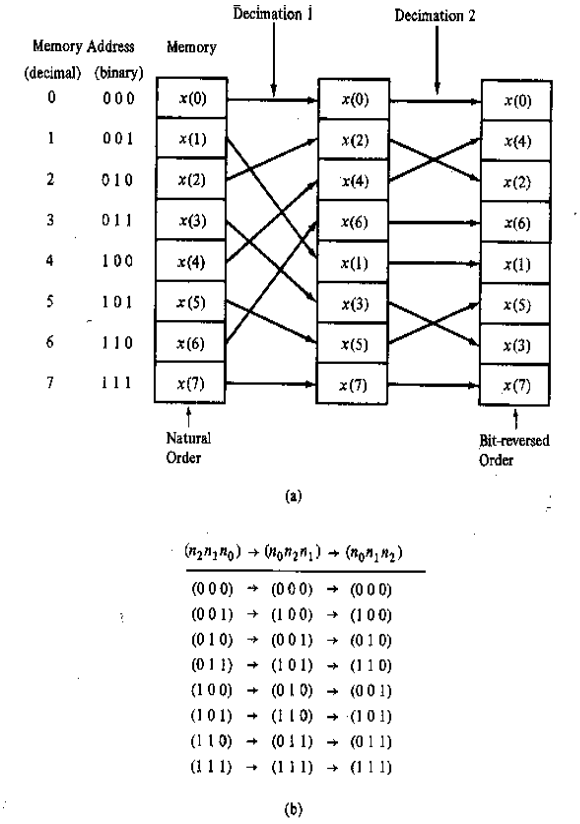

Figure -1.5: Shuffling of the data and bit reversal**.**

Another important radix-2 FFT algorithm, called the decimation-in-frequency algorithm, is obtained by using the divide-and-conquer approach. To derive the algorithm, we begin by splitting the DFT formula into two summations, one of which involves the sum over the first N/2 data points and the second sum involves the last N/2 data points. Thus we obtain

$$X(k) = \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn}$$
$$= \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + W_N^{Nk/2} \sum_{n=0}^{(N/2)-1} x\left(n + \frac{N}{2}\right) W_N^{kn}$$

Since $W_N^{kN/2} = (-1)^k$

$$X(k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + (-1)^k x\left(n + \frac{N}{2}\right) \right] W_N^{kn}$$

Now, let us split (decimate) X(k) into the even- and odd-numbered samples. Thus we obtain

$$X(2k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + x\left(n + \frac{N}{2}\right) \right], \qquad k = 0,1,\ldots,\frac{N}{2} - 1$$

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} \left\{ \left[x(n) - x\left(n + \frac{N}{2}\right) \right] \right\}, \qquad k = 0,1,\ldots,\frac{N}{2} - 1$$

where we have used the fact that $W_N^2 = W_{N/2}$

The computational procedure above can be repeated through decimation of the N/2-point DFTs X(2k) and X(2k+1). The entire process involves $v = \log_2 N$ stages of decimation, where each stage involves N/2 butterflies of the type shown in Figure2.7. Consequently, the computation of the N-point DFT via the decimation-in-frequency FFT requires $(N/2)\log_2 N$ complex multiplications and $N\log_2 N$ complex additions, just as in the decimation-in-time algorithm. For illustrative purposes, the eight-point decimation-in-frequency algorithm is given in Figure 1.8.

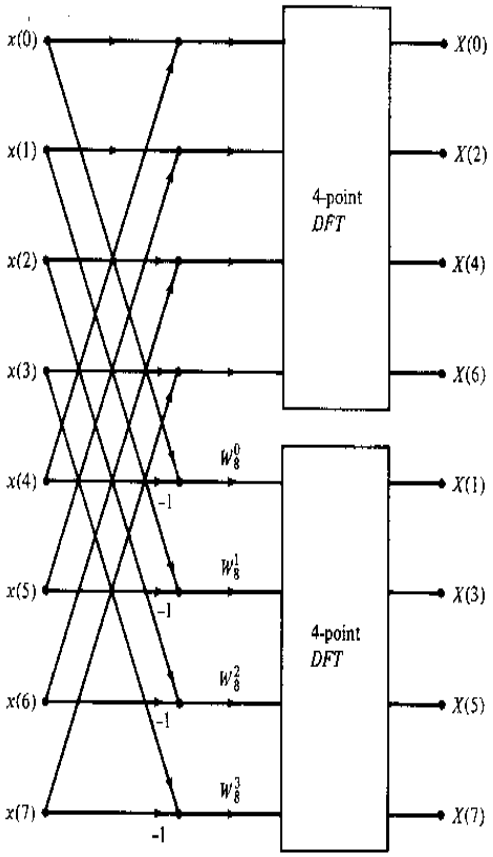

Figure-1.6: First stage of the decimation-in-frequency FFT algorithm.

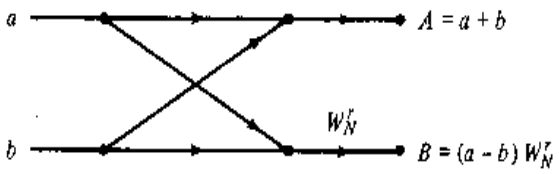

Figure-1.7: Basic butterfly computation in the decimation in frequency.

We observe from Figure TC.3.8 that the input data x(n) occurs in natural order, but the output DFT occurs in bit-reversed order.

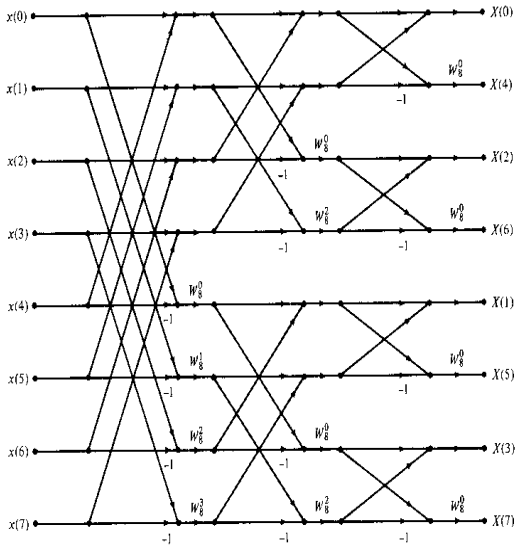

Figure-1.8 N=8-piont decimation-in-frequency FFT algorithm.

We also note that the computations are performed in place. However, it is possible to reconfigure the decimation-in-frequency algorithm so that the input sequence occurs in bit-reversed order while the output DFT occurs in normal order. Furthermore, if we abandon the requirement that the computations be done in place, it is also possible to have both the input data and the output DFT in normal order.

3. ENCRYPTION:

Encryption is the conversion of data into a form, called a cipher text that cannot be easily understood by unauthorized people. Decryption is the process of converting encrypted data back into its original form, so it can be understood.

The use of encryption/decryption is as old as the art of communication. In wartime, a cipher, often incorrectly called a code, can be employed to keep the enemy from obtaining the contents of transmissions. (Technically, a code is a means of representing a signal without the intent of keeping it secret; examples are Morse code and ASCII.) Simple ciphers include the substitution of letters for numbers, the rotation of letters in the alphabet, and the "scrambling" of voice signals by inverting the sideband frequencies. More complex ciphers work according to sophisticated computer algorithms that rearrange the data bits in digital signals .Encryption finds its use in many scenarios as follows:

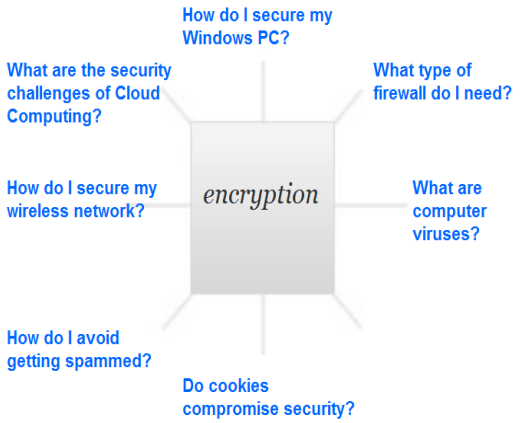

Figure -3.1: Uses of encryption methods

Encryption refers to algorithmic schemes that encode plain text into non-readable form or cipher text, providing privacy. The receiver of the encrypted text uses a "key" to decrypt the message, returning it to its original plain text form. The key is the trigger mechanism to the algorithm. Until the advent of the Internet, encryption was rarely used by the public, but was largely a military tool. Today, with online marketing, banking, healthcare and other services, even the average householder is aware of encryption. As more people realize the open nature of the Internet, email and instant messaging, encryption will undoubtedly become more popular. Without encryption, information passed on the Internet is not only available for virtually anyone to snag and read, but is often stored for years on servers that can change hands or become compromised in any number of ways. For all of these reasons encryption is a goal worth pursuing. In order to easily recover the contents of an encrypted signal, the correct decryption key is required. The key is an algorithm that undoes the work of the encryption algorithm. Alternatively, a computer can be used in an attempt to break the cipher. The more complex the encryption algorithm, the more difficult it becomes to eavesdrop on the communications without access to the key. Often there has been a need to protect information from 'prying eyes'. In the electronic age, information that could otherwise benefit or educate a group or individual can also be used against such groups or individuals. Industrial espionage among highly competitive businesses often requires that extensive security

measures be put into place. And, those who wish to exercise their personal freedom, outside of the oppressive nature of governments, may also wish to encrypt certain information to avoid suffering the penalties of going against the wishes of those who attempt to control. Still, the methods of data encryption and decryption are relatively straightforward, and easily mastered. I have been doing data encryption since my college days, when I used an encryption algorithm to store game programs and system information files on the university mini-computer, safe from 'prying eyes'. These were files that raised eyebrows amongst those who did not approve of such things, but were harmless. I was occasionally asked what this "rather large file" contained, and I once demonstrated the program that accessed it, but you needed a password to get to 'certain files' nonetheless. And, some files needed a separate encryption program to decipher them.

Encryption/decryption is especially important in wireless communications. This is because wireless circuits are easier to tap than their hard-wired counterparts. Nevertheless, encryption/decryption is a good idea when carrying out any kind of sensitive transaction, such as a credit-card purchase online, or the discussion of a company secret between different departments in the organization. The stronger the cipher -- that is, the harder it is for unauthorized people to break it -- the better, in general. However, as the strength of encryption/decryption increases, so does the cost. In recent years, a controversy has arisen over so-called strong encryption. This refers to ciphers that are essentially unbreakable without the decryption keys. While most companies and their customers view it as a means of keeping secrets and minimizing fraud, some governments view strong encryption as a potential vehicle by which terrorists might evade authorities. These governments, including that of the United States, want to set up a key-escrow arrangement. This means everyone who uses a cipher would be required to provide the government with a copy of the key. Decryption keys would be stored in a supposedly secure place, used only by authorities, and used only if backed up by a court order. Opponents of this scheme argue that criminals could hack into the key-escrow database and illegally obtain, steal, or alter the keys. Supporters claim that while this is a possibility, implementing the key escrow scheme would be better than doing nothing to prevent criminals from freely using encryption/decryption.

3.1 Types of Encryption Methods

There are three basic encryption methods: hashing, symmetric cryptography, and asymmetric cryptography. Each of these encryption methods have their own uses, advantages, and disadvantages. All three of these encryption methods use cryptography, or the science of scrambling data.

Cryptography is used to change readable text, called plaintext, into an unreadable secret format, called cipher text, using a process called encryption. Encrypting data provides additional benefits besides protecting the confidentiality of data. Other benefits include ensuring that messages have not been altered during transit and verifying the identity of the message sender. All these benefits can be realized by using basic encryption methods.

Hashing:

The first encryption method, called hashing, creates a unique fixed length signature of a group of data. Hashes are created with an algorithm, or hash function, and are used to compare sets of data. Since a hash is unique to a specific message, any changes to that message would result in a different hash, thereby alerting a user to potential tampering.

A hash algorithm, also known as a hash function, is a mathematical procedure used in computer programming to turn a large section of data into a smaller representational symbol, known as a hash key. The major use of hash algorithms occurs in large databases of information. Each collection of data is assigned a hash key, which is a short symbol or code that represents it. When a user needs to find that piece of data, he inputs the symbol or code and the computer displays the full data piece.

For hashing, as this process is called, to work it needs a hash function or hash algorithm. This tells the computer how to take the hash key and match it with a set of data it represents. Areas in the computer program known as slots or buckets store information and each key links to a specific slot or bucket. The entire process is contained within a hash table or hash map. This table records data and the matching keys that correspond to it. It then uses a hash algorithm to connect a key to a piece of data when the user requests it.

A key difference between a hash and the other two encryption methods is that once the data is encrypted, the process cannot be reversed or deciphered. This means that even if a potential attacker were able to obtain a hash, he would not be able to use a decryption method to discover the contents of the original message. Some common hashing algorithms are Message Digest 5 (MD5) and Secure Hashing Algorithm (SHA).

3.1.1 Symmetric Cryptography

Symmetric cryptography, which is also called private-key cryptography, is the second encryption method. The term "private key" comes from the fact that the key used to encrypt and decrypt data must remain secure because anyone with access to it can read the coded messages. This encryption method can be categorized as either a stream

cipher or a block cipher, depending upon the amount of data being encrypted or decrypted at a time. A stream cipher encrypts data one character at a time while a block cipher processes fixed chunks of data. Common symmetric encryption algorithms include Data Encryption Standard (DES), Advanced Encryption Standard (AES), International Data Encryption Algorithm (IDEA), and Blowfish. For symmetric key ciphers, there are basically two types: BLOCK CIPHERS, in which a fixed length block is encrypted, and STREAM CIPHERS, in which the data is encrypted one 'data unit' (typically 1 byte) at a time, in the same order it was received in. Fortunately, the simplest of all of the symmetric key 'stream cipher' methods is the TRANSLATION TABLE (or 'S table'), which should easily meet the performance requirements of even the most performance-intensive application that requires data to be encrypted. In a translation table, each 'chunk' of data (usually 1 byte) is used as an offset within one or more arrays, and the resulting 'translated' value is then written into the output stream. The encryption and decryption programs would each use a table that translates to and from the encrypted data. 80x86 CPU's have an instruction 'XLAT' that lends itself to this purpose.

While translation tables are very simple and fast, the down side is that once the translation table is known, the code is broken. Further, such a method is relatively straightforward for code breakers to decipher - such code methods have been used for years, even before the advent of the computer. Still, for general "unread ability" of encoded data, without adverse effects on performance, the 'translation table' method lends itself well. A modification to the 'translation table' uses 2 or more tables, based on the position of the bytes within the data stream, or on the data stream itself. Decoding becomes more complex, since you have to reverse the same process reliably. But, by the use of more than one translation table, especially when implemented in a 'pseudo-random' order, this adaptation makes code breaking relatively difficult. An example of this method might use translation table 'A' on all of the 'even' bytes, and translation table 'B' on all of the 'odd' bytes. Unless a potential code breaker knows that there are exactly 2 tables, even with both source and encrypted data available the deciphering process is relatively difficult.

Similar to using a translation table, 'data repositioning' lends itself to use by a computer, but takes considerably more time to accomplish. This type of cipher would be a trivial example of a block cipher. A buffer of data is read from the input, then the order of the bytes (or other 'chunk' size) is rearranged, and written 'out of order'. The decryption program then reads this back in, and puts them back 'in order'. Often such a method is best used in combination with one or more of the other encryption methods mentioned here, making it even more difficult for code breakers to determine how to decipher your encrypted data. As an example, consider an anagram. The letters are all there, but the order has been changed. Some anagrams are easier than others to decipher, but a well written anagram is a brain teaser nonetheless, especially if it's intentionally misleading.

High entropy data is difficult to extract information from, and the higher the entropy, the better the cipher. So, if you rotate the words or bytes within a data stream, using a method that involves multiple and variable direction and duration of rotation, in an easily reproducible pattern, you can quickly encode a stream of data with a method that can be nearly impossible to break. Further, if you use an 'XOR mask' in combination with this ('flipping' the bits in certain positions from 1 to 0, or 0 to 1) you end up making the code breaking process even more difficult. The best combination would also use 'pseudo random' effects, the easiest of which might involve a simple sequence like Fibonacci numbers, which can appear 'pseudo-random' after many iterations of 'modular' arithmetic (i.e. math that 'wraps around' after reaching a limit, like integer math on a computer). The Fibonacci sequence '1,1,2,3,5,...' is easily generated by adding the previous 2 numbers in the sequence to get the next. Doing modular arithmetic on the result and operating on multiple byte sequences (using a prime number of bytes for block rotation, as one example) would make the code breaker's job even more difficult, adding the 'pseudo-random' effect that is easily reproduced by your decryption program.

3.1.2 Asymmetric Cryptography

Asymmetric or public key, cryptography is the last encryption method. This type of cryptography uses two keys, a private key and a public key, to perform encryption and decryption. The use of two keys overcomes a major weakness in symmetric key cryptography in that a single key does not need to be securely managed among multiple users. In asymmetric cryptography, a public key is freely available to everyone while the private key remains with receiver of cipher text to decrypt messages. Algorithms that use public key cryptography include RSA and Diffie-Hellman.

The advantage of asymmetric over symmetric key encryption, where the same key is used to encrypt and decrypt a message, is that secure messages can be sent between two parties over a non-secure communication channel without initially sharing secret information. The disadvantages are that encryption and decryption is slow, and cipher text potentially may be hacked by a cryptographer given enough computing time and power. One very important feature of a good encryption scheme is the ability to specify a 'key' or 'password' of some kind, and have the encryption method alter itself such that each 'key' or 'password' produces a unique encrypted output, one that also requires a unique 'key' or 'password' to decrypt. This can either be a symmetric or asymmetric key. The popular 'PGP' public key encryption, and the 'RSA' encryption that it's based on, uses an 'asymmetrical' key, allowing you to share the 'public' encryption key with everyone, while keeping the 'private'

decryption key safe. The encryption key is significantly different from the decryption key, such that attempting to derive the private key from the public key involves too many hours of computing time to be practical. It would NOT be impossible, just highly unlikely, which is 'pretty good'. There are few operations in mathematics that are truly 'irreversible'. In nearly all cases, the commutative property or an 'inverse' operation applies. if an operation is performed on 'a', resulting in 'b', you can perform an equivalent operation on 'b' to get 'a'. In some cases you may get the absolute value (such as a square root), or the operation may be undefined (such as dividing by zero). However, it may be possible to base an encryption key on an algorithm such that you cannot perform a direct calculation to get the decryption key. An operation that would cause a division by zero would PREVENT a public key from being directly translated into a private key. As such, only 'trial and error' (otherwise known as a 'brute force' attack) would remain as a valid 'key cracking' method, and it would therefore require a significant amount of processing time to create the private key from the public key.

In the case of the RSA encryption algorithm, it uses very large prime numbers to generate the public key and the private key. Although it would be possible to factor out the public key to get the private key (a trivial matter once the 2 prime factors are known), the numbers are so large as to make it very impractical to do so. The encryption algorithm itself is ALSO very slow, which makes it impractical to use RSA to encrypt large data sets. So PGP (and other RSA-based encryption schemes) encrypt a symmetrical key using the public key, then encrypt the remainder of the data with a faster algorithm using the symmetrical key. The symmetrical itself key is randomly generated, so that the only (theoretical) way to get it would be by using the private key to decrypt the RSA-encrypted symmetrical key.

Example: Suppose you want to encrypt data (let's say this web page) with a key of 12345. Using your public key, you RSA-encrypt the 12345, and put that at the front of the data stream (possibly followed by a marker or preceded by a data length to distinguish it from the rest of the data). THEN, you follow the 'encrypted key' data with the encrypted web page text, encrypted using your favorite method and the key '12345'. Upon receipt, the decrypt program looks for (and finds) the encrypted key, uses the 'private key' to decrypt it, and gets back the '12345'. It then locates the beginning of the encrypted data stream, and applies the key '12345' to decrypt the data. The result: a very well protected data stream that is reliably and efficiently encrypted, transmitted, and decrypted.

4. LABVIEW INTRODUCTION

Lab VIEW (Laboratory Virtual Instrumentation Engineering Workbench) is a platform and development environment for a visual programming language from National Instruments. The graphical language is named "G". Lab VIEW is commonly used for data acquisition, instrument control, and industrial automation on a variety of platforms including Microsoft Windows, various flavors of UNIX, Linux, and Mac OS. The latest version of Lab VIEW is version 8.6.1, released in February of 2009.

Lab VIEW is a program development application, much like C or FORTRAN. Lab VIEW is, however, different from those applications in one important aspect. Other programming systems use text-based languages to create lines of code, while Lab VIEW uses a graphical programming language, G, to create programs in block diagram form.

Lab VIEW, like C or FORTRAN, is a general-purpose programming system with extensive libraries of functions for many programming tasks. Lab VIEW includes libraries for data acquisition, data analysis, data presentation, and data storage. A Lab VIEW program is called a virtual instrument (VI) because it’s appearance and operation can imitate an actual instrument.

It is specifically designed to take measurements, analyze data and present results to the user. It has such a versatile graphical user interface and is easy to program with, it is also ideal for simulations, presentation of ideas and general programming. Academic campuses worldwide use it to deliver project-based learning. Lab VIEW offers unrivaled integration with thousands of hardware devices and provides hundreds of built-in libraries for advanced analysis and data visualization. With the intuitive nature of the graphical programming environment, we can:

- Visualize and explore theoretical concepts through interactive simulations and real-world signals.
- Design projects in applications such as measurement, control, embedded, signal processing, and communication.
- Compute, simulate, and devise solutions to homework problems.

5. WORKING

The basic operation of the project designed can be illustrated as a process of encryption in encryptor module, transmitted through a communication medium to the decryptor module and the retrieval of the original text file at the output of decryptor module. The type of encryption method implemented in this application is a mixed type of Symmetric and Asymmetric Encryption. The various concepts holding as keys in the Encryption procedure one of them being the scrambling pattern needs to be implemented exactly in reversal order to decrypt the data which serves the purpose of symmetric method and availability of the users to login with different ids at encryptor and decryptor module serves the asymmetric method. Hence the encryption method implemented is of both the types. As described above the operation of the application is among the two vital modules

.The following diagram gives the illustration. The encryption and decryption process takes place in two modules of the project designed. The two modules in the project designed are:

1. Encryptor module
2. Decryptor module

5.1 Encryptor Module

The Encryptor module is the vital part of the application where the input text file is received for encryption. The Encryptor module designed for the application avail the cryptographer with user friendly and security options. The user friendly options provided by the programmer include the managing of the appropriate visibility of the login id options. The encryptor model provides the facility of creating a username and password in particular at the encryptor to proceed with the process of encryption. The Username and Password provides the security from any other person other than the assigned cryptographer to use the application.

The various user friendly options provided in using the login id mainly concerned to its appropriate timely visibility and ineligible logins using default null usernames and passwords. A sequence of coding has been done for proper operation of the username and password. Initially, the input for the username and the password and maintained at the null string or the empty space where the user could enter his username and password and the input text box, which shows the text present in the input text file which needs to be encrypted, is maintained empty and visible for the user. The usernames and passwords of eligible users registered are arranged as an array of usernames and passwords. Hence a case always arises where the zero index of the array of strings is an empty string and an unassigned user can login with empty strings and such a possibility is avoided with proper coding. After entering the username and password, an OK button is provided whose operation would decide the further proceedings into the application. When OK button is pressed or its state is changed after entering the login id details it further continues its operation of defining the user whether a valid one or an invalid one. Until the OK button is used even though after entering the details there would be no further action. Once the OK button is used the step executes, where it is checked whether the username matches with password of the concerned user. If an ineligible detail is entered the user will be provided his state as invalid user and once both the details match with database the application leads the user to further operation where the user is directed for the procedure of encryption. On the display the once a valid user enters the username, password and OK button disappear to avoid any confusion. Once the user login is done and procedure is completed a dialog showing encryption successful is shown and directed to the empty username and password blocks with OK button.

5.2 Flow Diagram of Encryption

The flow diagram of the encryption process in the encryptor module is:

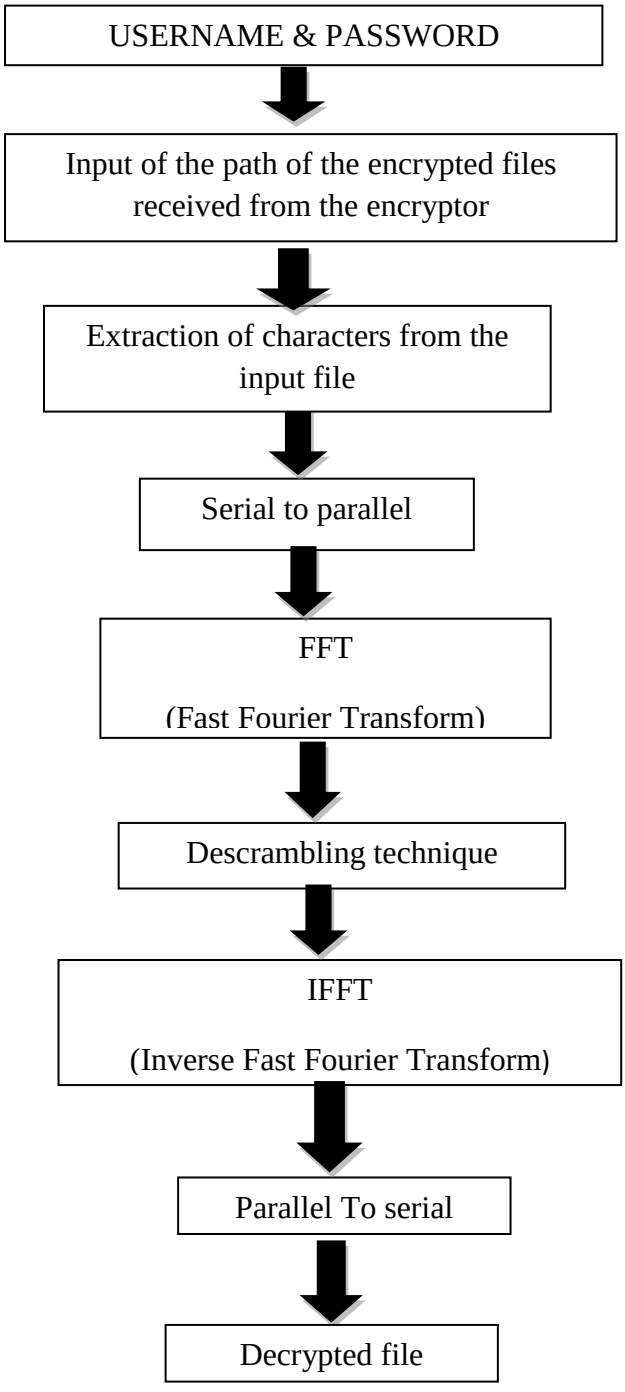

Figure - 5.1: Flow diagram of the encryption process

The above shows the flow diagram of the encryption process. Initially, the user needs to use the valid login id details to begin the application .In case of unknown details particular

user will be regarded an invalid user. Once, the user has logged in the application shows a browsing window to select a particular text file which needs to be encrypted and secured for further purpose. The user can browse for the path of the file and select it. Later, the application requests the user to create two empty files and provide its path in particular browsing windows to store the encrypted data after the encryption of the input text file. The characters from the input text file are extracted using the "Read from text file" function to encrypt. After the extraction of the characters the "string to array" function is used to assign every available alphabets and characters their respective ASCII codes.

5.3. ASCII Codes

The American Standard Code for Information Interchange is a character-encoding scheme originally based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that use text. Most modern character-encoding schemes are based on ASCII, though they support many additional characters. ASCII developed from telegraphic codes. Its first commercial use was as a seven-bit teleprinter code promoted by Bell data services. Work on the ASCII standard began on October 6, 1960, with the first meeting of the American Standards Association's (ASA) X3.2 subcommittee. The first edition of the standard was published during 1963, a major revision during 1967, and the most recent update during 1986. Compared to earlier telegraph codes, the proposed Bell code and ASCII were both ordered for more convenient sorting (i.e., alphabetization) of lists and added features for devices other than teleprinters.

After the conversion of characters into ASCII codes .the series of data available needs to be converted into a parallel blocks OK data, each block containing 8 bits of data as the Fast Fourier Transform used for domain conversion is an algorithm implemented for input of 8 bits. For such a specific serial-to-parallel conversion a serial to parallel converter is designed as ,All the data available is placed in a "for Loop" whose value of N fixed to the value of the total number of elements present in the input file divided by eight so that those many number of blocks of 8 bits are formed. The number elements present in the input file is known using "string length" function and a "divide" with constant value eight to give the value to N. Later the array function "Array Subset" is used to form the arrays of 8 bits and input for length parameter in this function is given by the iteration value and a "multiply" function with constant eight. Then an "Array to Cluster" is used to give input to the FFT algorithm which converts domain from time domain to frequency domain. The outputs of the FFT conversion are scrambled using scrambling technique and the scrambled values undergo the Inverse Fast Fourier Transform to give encrypted time domain values .But, the outputs obtained are in complex form and thus when they are directly given to the decryptor section the application would round the complex value of to its real value leaving imaginary values. Hence real and imaginary values are obtained from the output complex values and all the available real values are stored in one file and all the imaginary values are stored in another file so, that no information is lost.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Table -1.1: Chart for ASCII codes

5.4. Labview Implementation of Encryptor

Front Panel of Encryptor

The Front panel of the Encryptor module shows the various security options of login id and the input text space where the input texts file data. The empty space in the input text option shown displays the data of the text file which is encrypted for security purpose. During the execution of the code after entering login details a request for input text file and empty files to store the encrypted data. The coding of the encryptor module is done in such a user friendly manner with appropriate visibility of display options and proper message inputs to the user.

Block Diagram of Encryptor

The block diagram of the encryptor module shows the sequence of coding occurring during the operation of the application. The following block diagram shown in the figure includes the extraction of characters from the input text file, Conversion of obtained data from serial to parallel blocks of data, scrambling block and saving of the encrypted data in two files. One of the file possessing the real part of the output values and the other file possessing the imaginary parts of the outputs. A complex to polar conversion function is used to obtain the particular data in two files. In the figure encrypted part1 and encrypted part2 in the last two sequences shows the outputs of the Encryptor module in real and imaginary parts.

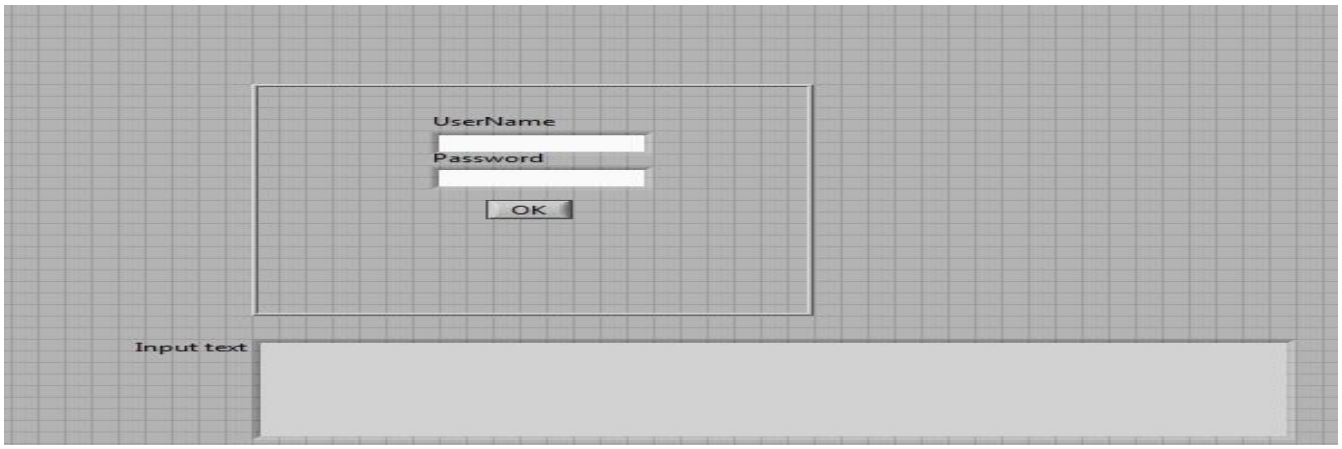

Figure -5.2: :Front Panel of the Encryptor module

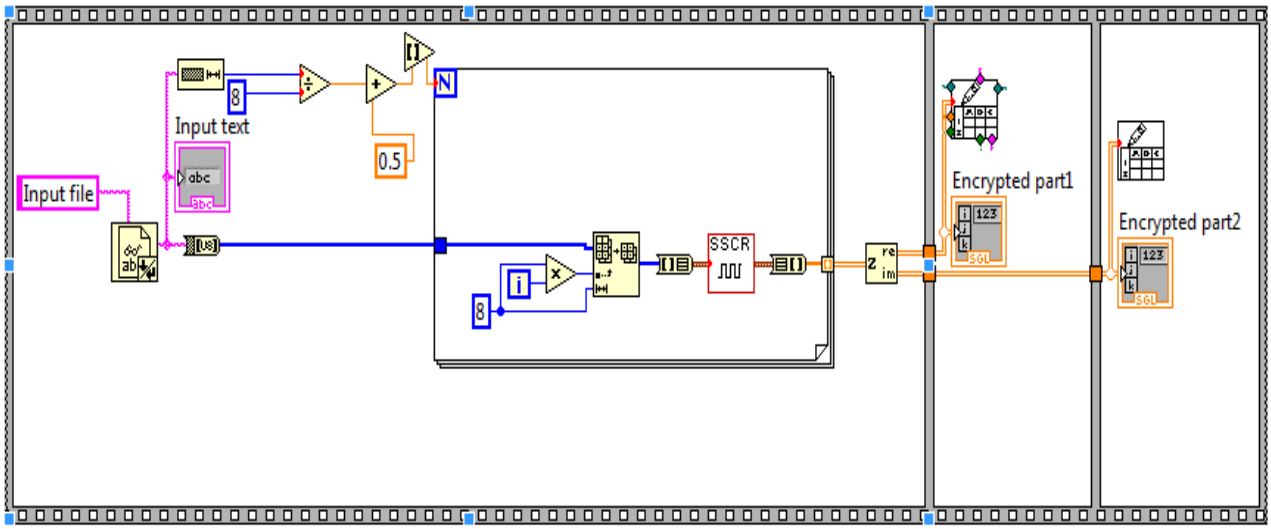

Figure 5.3: Block Diagram of the Encryptor module

As mentioned earlier in the introduction to the lab VIEW usage, the programming of any code would clear and illustrated for the other users using it when the code broken into individual codes based on their function and designed as individual VI for each broken code .the individual code designed is used as subVIs in the other VI which would be easy for understanding and illustration. In Encryptor module certain functions are broken and VI is created. These VI's created are added to actual VI. The subVIs added to the actual Encryptor module VI are:

1. Design for creation of Username and Password.
2. Design for scrambling pattern with FFT and IFFT subVIs.
3. Design for 8-bit FFT algorithm
4. Design for 8-bit IFFT algorithm

5.4.1) Block Diagram for Designing Username And Password In Encryptor Module:

The following figure gives the designing of username and password created for an Encryptor module. The Username and password functions are designed possessing various user friendly options. The various user friendly options provided in using the login id mainly concerned to its appropriate timely visibility and ineligible logins using default null usernames and passwords. A sequence of coding has been done for proper operation of the username and password.

Initially, the input for the username and the password and maintained at the null string or the empty space where the user could enter his username and password and the input text box, which shows the text present in the input text file which needs to be encrypted, is maintained empty and visible for the user. The usernames and passwords of eligible users registered are arranged as an array of usernames and passwords. Hence a case always arises where the zero index of the array of strings is an empty string and an unassigned user can login with empty strings and such a possibility is avoided with proper coding. After entering the username and password, an OK button is provided whose operation would decide the further proceedings into the application. When OK button is pressed or its state is changed after entering the login id details it further continues its operation of defining the user whether a valid one or an invalid one. Until the OK button is used even though after entering the details there would be no further action. Once the OK button is used the step executes, where it is checked whether the username matches with password of In programming an efficient and secure Login operation various functions are implemented in sequence of steps. Initially, to display the username and password empty spaces "local variable" for each of the username, password and input text/output text space are created and an empty string is given as input so that they display nothing. To enable the visibility of the Username, Password and OK button their "visible" options are created and set to "true" and the visible option of text space is disappeared by setting it to "false" until the login details are provided. In case a user tries to enter an empty string as username and password no action takes place. This is implemented by using a "while Loop" and an output of "invalid user" pops out. For the convenience an "OK button" is created which needs to be used after entering the login details. In case of no operation of OK button there would be no action as output. After entry of the username and password the details are verified with the database created using "search 1D Array", input string constants and "Index Array Function". When the details match a message "Valid User" is given and the login options disappear, this done when visible of these options is given "false". Later the text space appears when a "true" is given to its visible option. After the Encryption process the "Encryption Successful" message is given.

5.4.2: Block Diagram for Designing Scrambling Pattern

The scrambling pattern is one of the important factor for security of data during transmission. The scrambling pattern created by the cryptographer acts as the key which stores data. The method of implementing scrambling pattern is changing the order of the variables in a desired fashion, different from the original order. By changing the position of the occurrence of the variables is changed then they are said to be scrambled. Thus in block diagram implementation the scrambling pattern is designed using the "Unbundle" function.

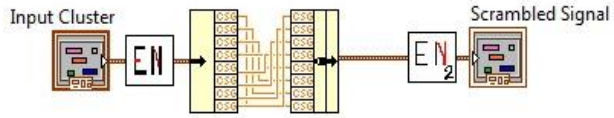

Figure -5.4: Design for scrambling pattern with FFT and IFFT subVIs

5.4.3 Block Diagram for Designing A 8-Bit Fft Algorithm

Therefore in Encryption process, the different type of values obtained from the character to ASCII code conversion and serial to parallel conversion are considered as a "Cluster". A cluster is set of values of different types similar to arrays. These clusters formed are of 8-bits. Thus for domain conversion an 8-bit FFT algorithm needs to be implemented to convert the data from time domain to frequency domain. the 8-bit FFT butterfly diagram is implemented using Lab VIEW functions. Therefore "Bundle", "Unbundle","Multiply", "Add" and "Subtract" functions are used. The following figure gives the block diagram of 8-bit FFT algorithm.

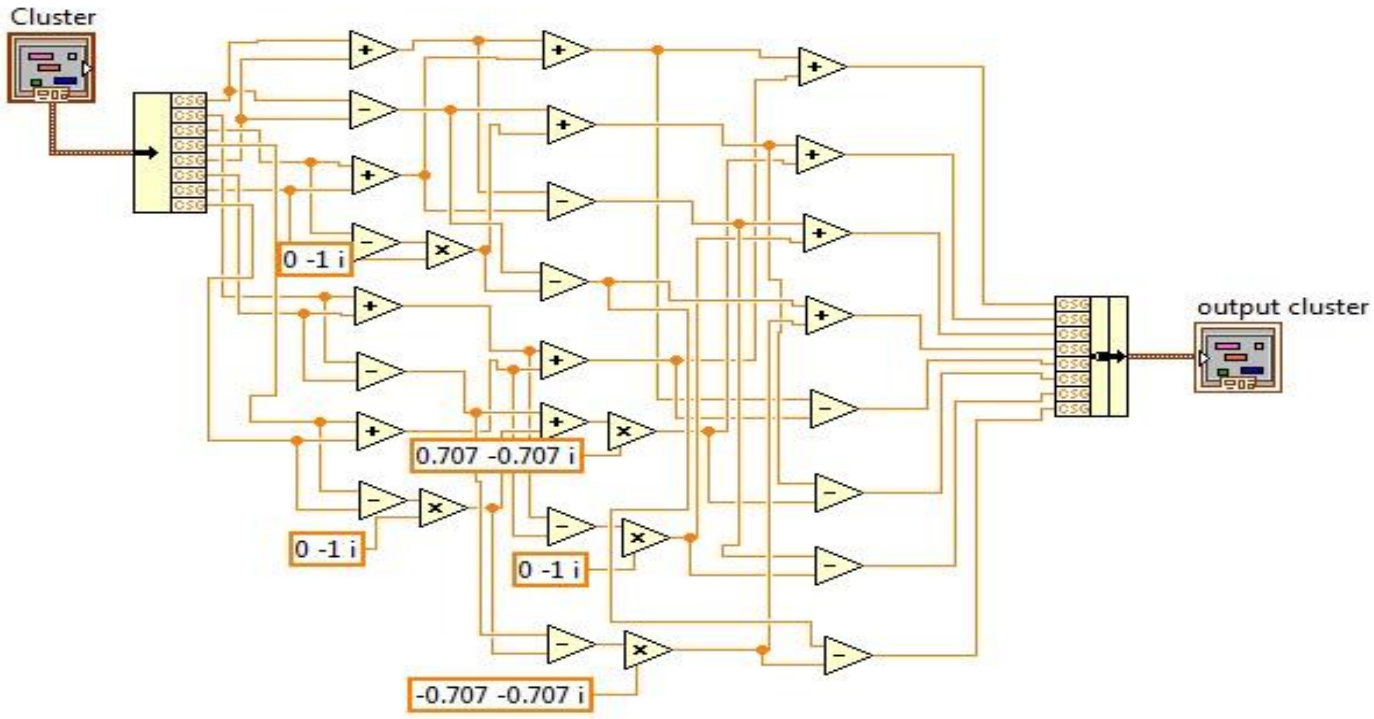

Figure 5.5: Design for 8-bit FFT algorithm

5.4.4 Block Diagram for Designing 8-Bit Ifft

Algorithm:

In encryption, after the scrambling the data needs to undergo domain conversion to its actual Time domain. Thus, an 8-bit IFFT is used to convert the frequency domain data to time domain data. The following figure gives the implementation of an 8-bit IFFT.

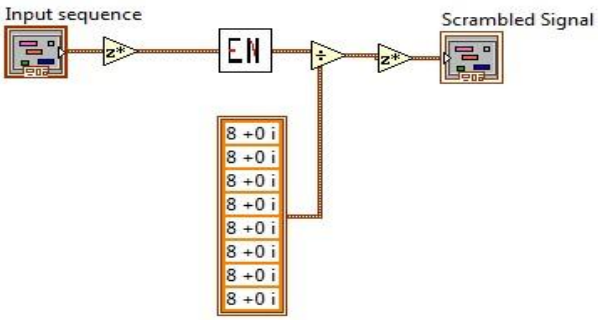

Figure 5.6: Design for 8-bit IFFT algorithm

5.5.1 Decryptor Module:

The Decryptor module is the other vital part of the application where the encrypted text files are received from encryptor. The Decryptor module designed for the application avail the cryptographer with user friendly and security options. The user friendly options provided by the programmer include the managing of the appropriate visibility of the login id options. The Decryptor model provides the facility of creating a username and password in particular at the Decryptor similar to that of encryptor. The Username and Password provides the security from any other person other than the assigned cryptographer to use the application. The various user friendly options are provided in decryptor as similar to that of encryptor.

There is a possibility that a user different from that at the encryptor can login with his/her details present in data base to conduct the process of decryption. Thus different but eligible users can use the application at the encryptor and decryptor ends. This adds the feature of Asymmetric encryption to the application.

The following figure gives the flow chart of the decryption process .Initially, once the login details are verified the request

for the encrypted files from the encryptor displays and after the paths are specified the real and imaginary data from the respective files is extracted. The real and imaginary data obtained is formed into a complex data which could be used for further process. The complex data obtained is let to undergo a domain conversion from time domain to frequency domain using an 8-bit FFT algorithm. The algorithm gives an array of blocks of 8-bit data. Elements in each block undergo scrambling similar to that of the encryptor. Thus the knowledge pertaining to the scrambling pattern should be possessed by respective cryptographers. Thus the application designed is a type of symmetric encryption along with the asymmetric encryption. After the scrambling the data undergoes a domain conversion from frequency domain to time domain using a IFFT algorithm.

Thus an 8-bit IFFT is implemented similar to that of encryptor. The IFFT gives parallel data of complex values containing only real values which are approximately equal to the ASCII values of the respective text characters. This parallel data is converted to serial data using a parallel to serial converter. In the implementation the parallel to serial converter implemented by using a concatenation function. In coding, the values are converted to charades using the "unsigned array to string" function and the shift registers to store the previous strings and "concatenate strings" for concatenation of to strings to for the whole paragraph of the text file. At the end a "Write to text file" function is used to write the characters into a file.

5.5.2. Labview Implementation of Decryptor Module:

Front Panel of Decryptor Module:

The Front panel of the Decryptor module shows the various security options of login id and the input text space where the input text file data. The empty space in the output text option shown displays the data of the text file which is encrypted for security purpose. During the execution of the code after entering login details a request for encrypted text files and an empty file to store the retrieved data. The coding of the Decryptor module is done in such a user friendly manner with appropriate visibility of display options and proper message inputs to the user.

Block Diagram of Decryptor Module:

The block diagram of the decryptor module shows the sequence of coding occurring during the operation of the application. The following block diagram shown in the figure includes the extraction of characters from the input encrypted text files, scrambling block and concatenation of the parallel strings obtained. After the process of concatenation the strings written into the empty file which was created in beginning of the application

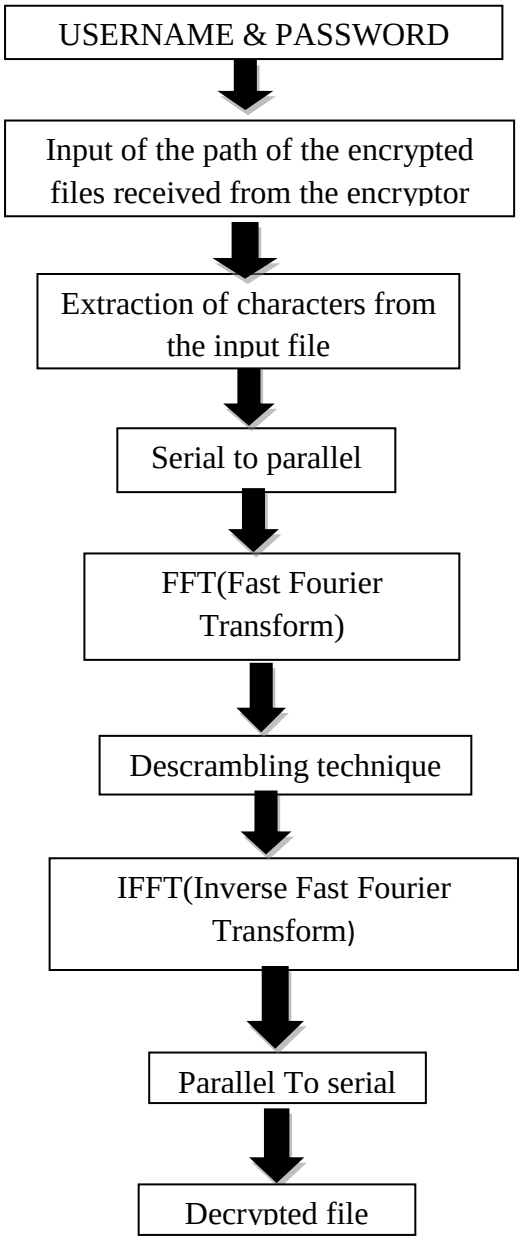

Figure 5.7: Design for 8-bit IFFT algorithm

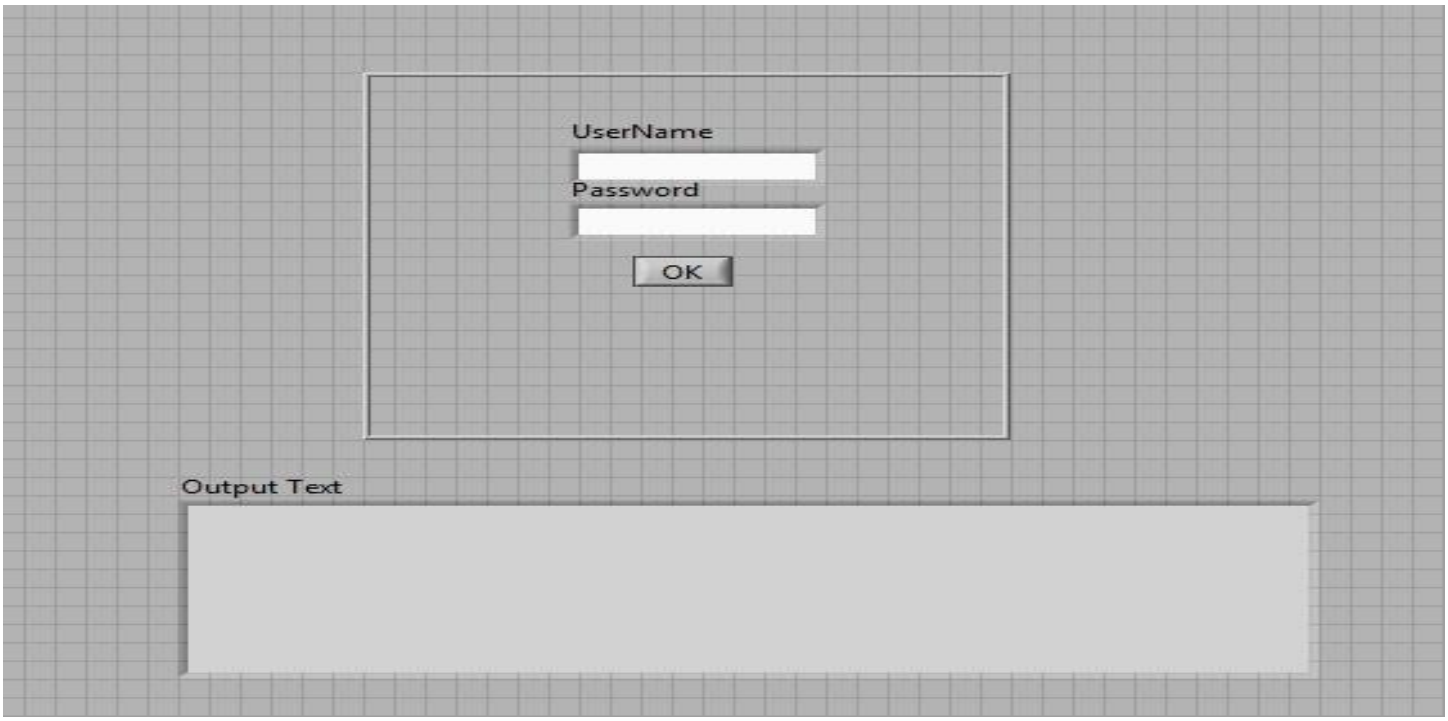

Figure -5.8: Front Panel for Decryptor Module

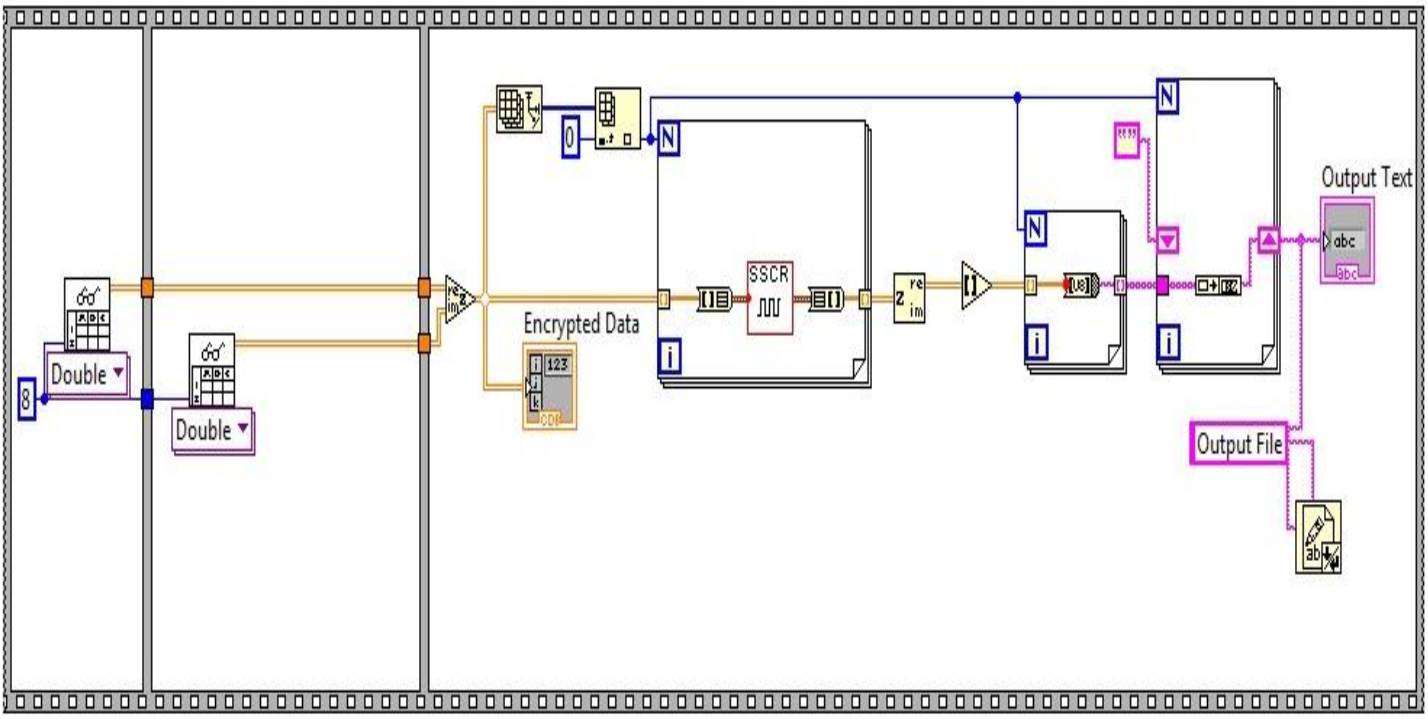

Figure -5.9.: Block Diagram for Decryptor Module

RESULTS:

a)ENCRYPTION OUTPUT:

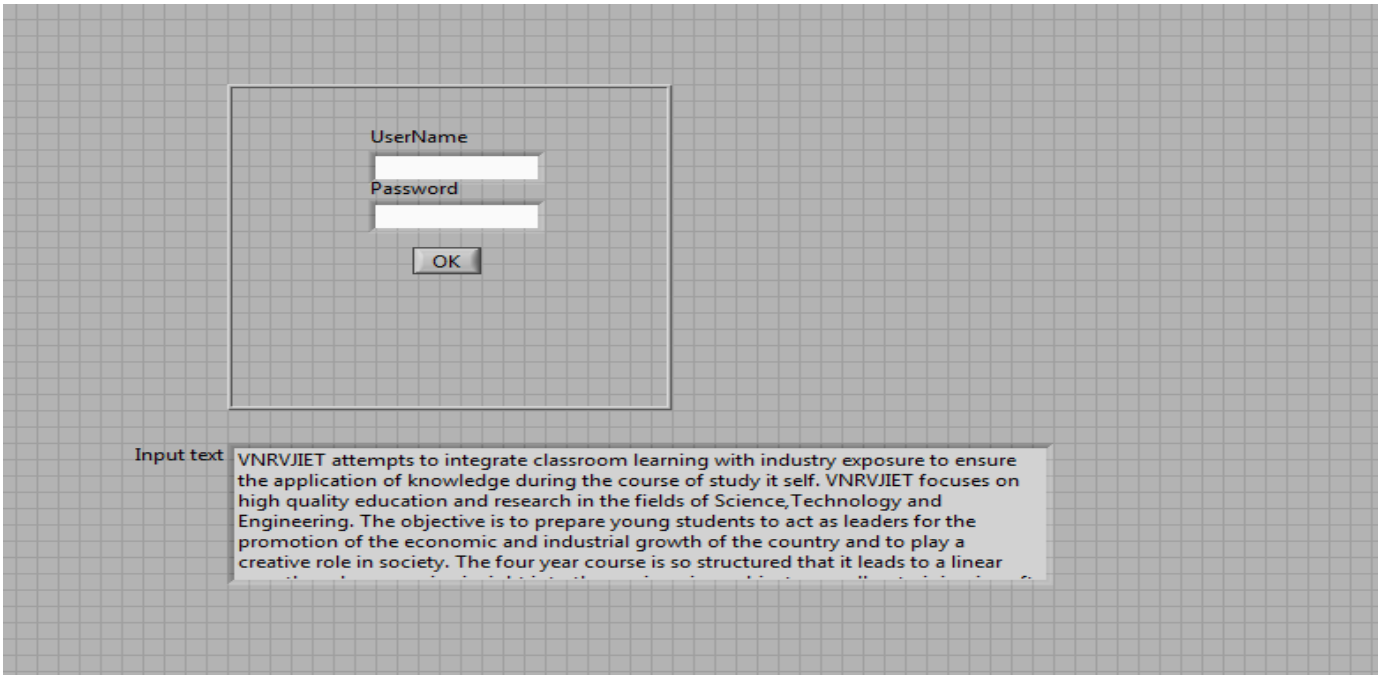

Figure -6.1: Block Diagram for Decryptor Module

b)DECRYPTION OUTPUT:

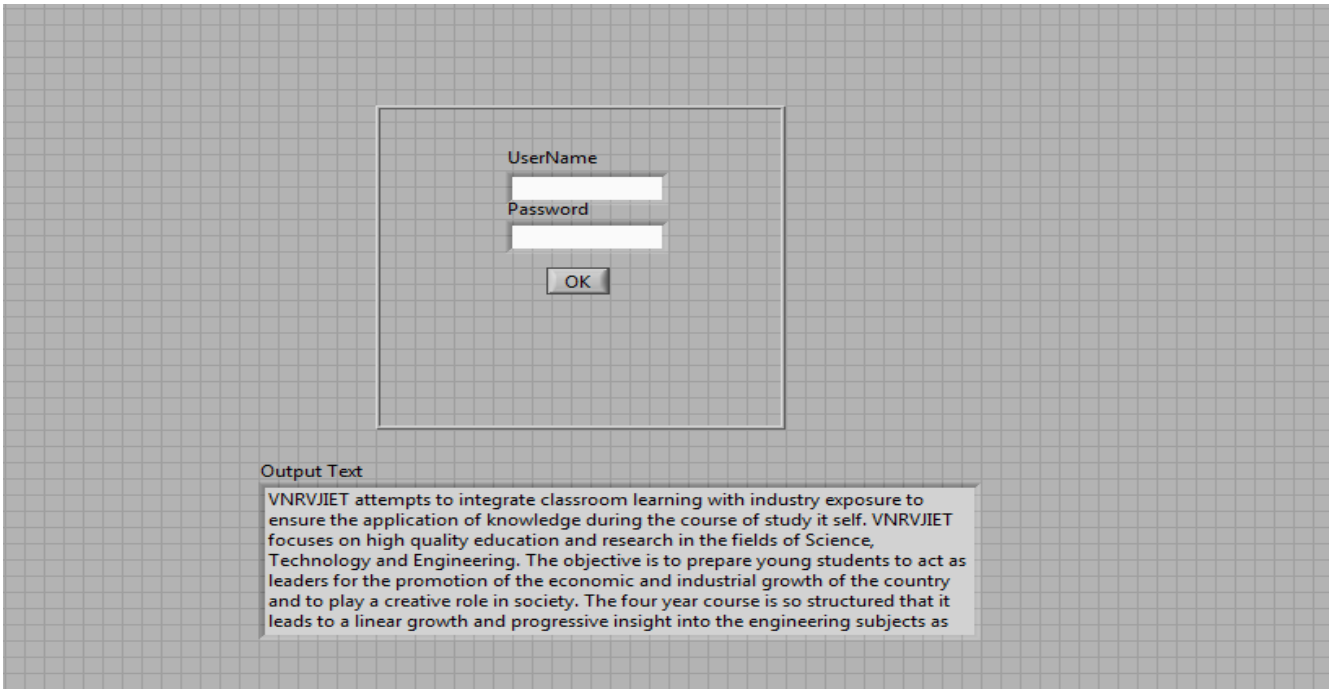

Figure -6.2: Block Diagram for Decryptor Module

FUTURE ASPECTS:

The application possesses immense scope for further development. The development relies on the factors providing security for the transmission. The application can be further designed for word documents and other types of text files (pdf, word document, etc).the security aspect provided by the scrambling pattern can be enhanced by designing a higher bit Fast Fourier Transform (FFT) than 8-bit which needs a higher bit scrambling pattern. At present there is possibility of 8! i.e. 40320 scrambling patterns for example if a 16-bit scrambling pattern is developed 20,922,789,888,000 which make it highly not possible to crack. for using a 16-bit scrambling pattern a 16 bit serial to parallel conversion should be developed. In the present application the encrypted data is split into two files for real and imaginary values thus the data can be split into more than two files this feature adds to the security i.e. in case a hacker is in possession of one of the file cannot crack the original file thus when number of split files increase the security. The proposed Encryption technique can be implemented for audio, image etc inputs.

CONCLUSIONS

The application designed provides a very high security in transmission of a text file. This application can be used as an induced encryptor in programming scenarios to secure vital codes and secure transmission in research departments. the high level of security is provided by the scrambling patterns, user logins, domain conversion etc. the future scope of this application relies the development of higher order FFT algorithms which proportionally increases the possible number of scrambling patterns for higher security.

BIOGRAPHIES:

[1] http://www.honeypage.com/SPEECH_ENCRYPTION_AND_DECRYPTION_USING_DSP_PROCESSOR.html

[2] Fast Fourier transform based speech encryption System - S. Sridharan, E. Dawson, Goldburg .IEEE PROCEEDINGS-I, Vol. 138, No. 3, JUNE 1991

[3] 'LabVIEW for Everyone'-Jeffrey Travis, Jim Kring

[4] Encryption using Fast Fourier Transform Techniques – S. Sridharan, E. Dawson & J. O' Sullivan

[5] LabVIEW for Digital Signal processing and digital communication – Cory L. Clark

[6] Digital Signal Processing – John G. Prokais

[7] "Cryptanalysis of frequency domain" - B. Goldburg, S. Sridharan and E, Dawson, Volume:140,Issue:4,Publication Year: 1993 , Page(s): 235 - 239

[8] "The handbook of real-time Fourier transforms," - W. Smith and J. Smith,.

[9] "Fourier transforms: An introduction for engineers,"- R.M. Gray and J.W. Goodman