Implementation of AES algorithm using GF (3^n) modulo system

A Major Project Report

Submitted in the partial fulfilment of the requirements for

The award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

M.V.S.N.Uzval (13003119)

P. Sai Reddy (13007543)

G.V. Krishna Chaitanya (13003274)

Under the supervision of

Dr. K. Rajasekhar

Professor

Department of Computer Science and Engineering

K L University, Green Fields,

Vaddeswaram- 522502, Guntur (Dist), Andhra Pradesh, India.

2016- 2017

K L University

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Declaration

The Major Project Report entitled "**Implementation of AES algorithm using GF (3^n) modulo system**" is a record of bonafide work of **M.V.S.N. Uzval (13003119), P. Sai Reddy (13007543), G.V. Krishna Chaitanya (13003274)** submitted in partial fulfilment for the award of B.Tech in Computer Science and Engineering to the K L University. The results embodied in this report have not been copied from any other departments/University/Institute.

M.V.S.N. Uzval (13003119)

P. Sai Reddy (13007543)

G.V. Krishna Chaitanya (13003274)

K L University

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is to certify that the Major Project Report entitled "**Implementation of AES algorithm using GF (3^n) modulo system**" is being submitted by **M.V.S.N.Uzval (13003119), P. Sai Reddy (13007543), G.V.Krishna Chaitanya (13003274)** submitted in partial fulfilment for the award of B.Tech in Computer Science and Engineering to the K L University is a record of bonafide work carried out under our guidance and supervision.

The results embodied in this report have not been copied from any other departments/University/Institute.

Signature of the Supervisor
Dr. K. Rajasekhar
Professor

Signature of the HOD
Dr. V.Srikanth
Professor

External Examiner

ACKNOWLEDGEMENTS

We are very thankful to our project guide **Dr. K. Rajasekhar, Professor** for his continuous support and encouragement in completing the Project work. Without his help the project couldn't have been completed.

We take immense pleasure in expressing our gratitude to our head of Department **Dr.V.Srikanth, Professor** for his priceless words of encouragement without which this would have been a herculean task.

We express our heart full gratitude to our President **Dr. Koneru Satyanarayana,** Chancellor **Dr.M.Ramamoorty,** Vice Chancellor **Dr.L.S.S Reddy,** Pro-Vice Chancellor **Dr. A.V.S.Prasad** and our beloved Principal **Dr. Anand Kumar** for providing infrastructure facilities in pursuing this project.

Last but not the least, we thank all Teaching and Non-Teaching Staff of our department and especially my classmates and my friends for their support in the completion of our project work.

M.V.S.N. Uzval (13003119)

P. Sai Reddy (13007543)

G.V. Krishna Chaitanya (13003274)

ABSTRACT

Generally there are unbounded numbers of ways in number system. But machines count them in two's whereas humans count them in tens. It is already proved that for the minimum number of computations to be available for a number, a base-e number system is chosen. As the integer value of e is closest to 3, we work on the number framework with base 3. In general there exists a system which can convert a ternary number into binary and then binary to polynomial form and then send the encrypted data to the collector of data. In this paper, we discuss an approach which can convert a ternary number directly into polynomial form, skipping the step of conversion to intermediate binary form. We implement AES algorithm using the proposed ternary system as a replacement for $GF(2^n)$.

Index Terms — Encryption, Ternary numbers, Mix Columns.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER - 1

INTRODUCTION

The ternary numeral system (also called base-3) has three as its base. Analogous to a bit, a ternary digit is a trit (trinary digit). One trit is equivalent to $\log_2 3$ (about 1.58496) bits of information. Although ternary most often refers to a system in which the three digits 0, 1, and 2 are all non-negative numbers, the adjective also lends its name to the balanced ternary system, comprising the digits -1, 0 and +1, used in comparison logic and ternary computers.

Representations of integer numbers in ternary do not get uncomfortably lengthy as quickly as in binary. For example, decimal 365 corresponds to binary 101101101 (9 digits) and to ternary 111112 (six digits). However, they are still far less compact than the corresponding representations in bases such as decimal.

The counting method of base-3 involves only the digits 0,1,2. Despite ternary, the greater part regularly alludes with an arrangement in which those three digits 0, 1, 2 are all non-negative numbers, the modifier likewise lends its sake of the adjusted ternary system, utilized within examination rationale and ternary workstations. These ternary numbers are used in number of substantial applications and in extensive variety of problems in mathematics which includes problems on weighting. But there is a proved statement that " No balanced ternary notation exist in the system ". Instead of using 0,1,2 this system uses digits 0,1,-1. The ternary system for a few decimal numbers is as follows

1	1	11	102	21	210
2	2	12	110	22	211
3	10	13	111	23	212
4	11	14	112	24	220
5	12	15	120	25	221
6	20	16	121	26	222
7	21	17	122	27	1000

8	22	18	200	28	1001
9	100	19	201	29	1002
10	101	20	202	30	1010

1.1 Arithmetic operations

1.1.1 Addition: In the addition operation of ternary numbers, the operation is performed by using the addition table which is available and is as follows

0	1	2
1	2	10
2	10	11

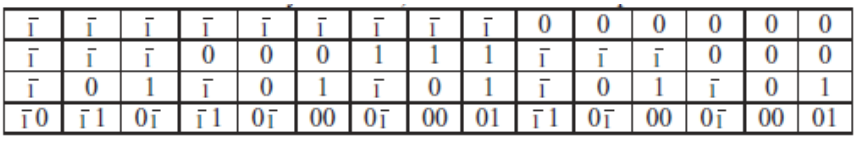

Figure 1 Pair of Trits to be added & carry bits

The column in the above table represents a pair of trits to be added and a carry bit. As it is base-3 number system the aggregate number of possible combinations of columns that are possible would be 27 (3^3).

1.1.2 Subtraction:

The subtraction operation of the ternary number system is same as the addition operation and involves the placement of negation symbol before the addition operation.

1.1.3 Multiplication:

For multiplication we store multiplicand in a register BR, say, and Multiplier in register QR, say. Initially, we assume that product is zero. This is known as the partial product, where a partial product is obtained by multiplying the multiplicand with one trit of the multiplier. In simple multiplication, if the trit of the multiplier is 1(1) then multiplicand is added (subtracted) with the partial product to generate a new partial product. Now the next trit of the multiplier is multiplied with multiplicand and the product is shifted by

one trit to the left and added with the partial product to generate a new partial product. But in case of hardware multiplication (using registers), instead of shifting the (multiplicand × c) (where c is a trit of the multiplier, having value 0 or 1 or 1) to the left we shift the partial product one trit to the right. We use the term Ashr to indicate arithmetic shift right. The multiplication algorithm is shown in figure. This operation has been defined for trits. This operation can likewise be performed by using the multiplication matrix which is as follows.

	1	2
1	1	2
2	2	11

The algorithms for multiplication operation in the ternary number system are as follows

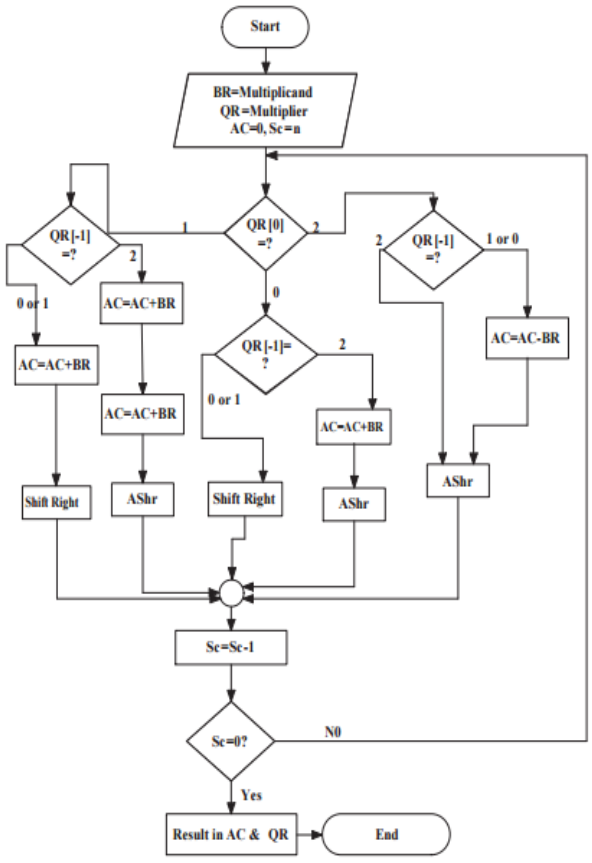

Figure 2 Algorithm for Multiplication operation

In the multiplication operation, multiplicand and multiplier are taken as inputs with a carry as another variable. We consider the product of multiplier and multiplicand initially as 0 , which is considered as partial product. In the multiplication algorithm the

incomplete item is included with the negation of multiplier and multiplicand to get a whole new partial product. Then, the negation of next multiplier is duplicated with the multiplicand and product is thus moved by one trit to one side and included with the fractional product. The same process is rehashed till all the multipliers are completed.

In general, the multiplication is characterized as multiplication with 1 is no change, multiplication with 2 implies shifting to the left and multiplication with 3 implies moving to left and then performing XOR operation with introductory unshifted esteem. In the wake of moving all the values are combined to obtain the result.

1.2 Polynomial

A polynomial is an expression which involves sum of powers in one or more variables multiplied by coefficients. A polynomial in one variable i.e. a univariate polynomial with constant coefficients is given by

$$a_n x^n + \ldots + a_2 x^2 + a_1 x + a_0$$

The monomials are nothing but the terms which are individual, whereas the products of the form $x_1^{a1} \ldots x_n^{an}$ in the multivariate case, i.e., with the coefficients omitted, are defined as terms. Subsequently, the term "monomial" is also used to mean polynomial summands without their coefficients, and in some older works, the monomial and term definitions are altered.

The highest power in a univariate polynomial is called its order, or degree itself sometimes.

Any polynomial P(X) with $P(0) \neq 0$ can be shown as

$$P(x) = P(0)\, \pi\, (1 - x/p)$$

where the result i.e product takes over the roots ρ of $P(\rho)=0$ and also, it is comprehended that different roots are checked with variety.

A polynomial in two variables (i.e., a bivariate polynomial) with constant coefficients is

given by

$$a_{nm}\, x^n\, y^m + \ldots. + a_{22}\, x^2\, y^2 + a_{21}\, x^2\, y + a_{12}\, x\, y^2 + a_{11}\, x\, y + a_{10}\, x + a_{01}\, y + a_{00}$$

The sum of two polynomials is nothing but the adding the coefficients which share the same powers of variables (i.e., the same terms) so, for example,

$$(a_2\, x^2 + a_1\, x + a_0) + (b_1\, x + b_0) = a_2\, x^2 + (a_1 + b_1)\, x + (a_0 + b_0)$$

and has order less than (in the case of cancellation of leading terms) or equal to the maximum order of the original two polynomials. Similarly, the product of two polynomials is obtained by multiplying term by term and combining the results, For example,

$$(a_2\, x^2 + a_1\, x + a_0) (b_1\, x + b_0) = a\, x^2 (b_1 x + b_0) + a_1 x (b_1 x + b_0) + a_0 (b_1 x + b_0)$$

$$= a_2 b_1 x^3 + (a_2 b_0 + a_1 b_1)\, x^2 + (a_1 b_0 + a_0 b_1) x + a_0 b_0$$

and has order equal to the sum of the orders of the two original polynomials.

1.3 Galois Field

In mathematics, more specifically in abstract algebra, Galois theory, named after Évariste Galois, provides a connection between field theory and group theory. Using Galois theory, certain problems in field theory can be reduced to group theory, which is, in some sense, simpler and better understood.

Originally, Galois used permutation groups to describe how the various roots of a given polynomial equation are related to each other. The modern approach to Galois theory, developed by Richard Dedekind, Leopold Kronecker and Emil Artin, among others, involves studying automorphisms of field extensions.

There are a number of different infinite fields, including the rational numbers (fractions), the real numbers (all decimal expansions), and the complex numbers. Cryptography focuses on finite fields. It turns out that for any prime integer p and any integer n greater than or equal to 1, there is a unique field with p^n elements in it, denoted $GF(p^n)$. (The ”GF” stands for “Galois Field”, named after the brilliant young French mathematician who discovered them.)

Here “unique” means that any two fields with the same number of elements must be essentially the same, except perhaps for giving the elements of the field different names.

In case n is equal to 1, the field is just the integers mod p, in which addition and multiplication are just the ordinary versions followed by taking the remainder on division by p. The only difficult part of this field is finding the multiplicative inverse of an element, that is, given a non-zero element a in Z_p, finding a^{-1}. This is the same as finding a b such that a*b % p = 1. This calculation can be done with the extended Euclidean algorithm, as is explained elsewhere in these notes.

A field which contains a definitely finite number of elements in it is called as Galois Field. Generally, Galois Field is applied on different operations of numbers such as addition, subtraction, multiplication and division. This Galois field is used in translating computer data as they are represented in binary, ternary formats i.e the computer data only contains 0,1 as these two are the only components in the Galois field.

The representation of Galois Field is as follows

$$GF(p^n)$$

Where,

p^n denotes the order of the field,

p denotes the characteristic of the field

The Galois field is used in encryption and decryption because each byte in the data are considered as a vector in the defined finite field. So, by using arithmetic it is very easy to encrypt and decrypt the data and is easily manipulable. Before the step of data encryption the data must be first arranged into a set of columns. The AES algorithm uses Mix Columns in which the new columns are created by reinstate the old columns with a definite key. In this step, the operations on the data in performed.

Before data scrambling and encryption can begin, the data must first be arranged in a state or a matrix of bytes. The algorithm for Advanced Encryption Standard (AES) consists of smaller, sub-algorithms namely SubBytes, ShiftRows, MixColumns, and AddRoundKey, where each method will be explained in details below. Note that the following explanation

only applies to AES with 128-bit key. The algorithm for other variations such as AES with 192-bit and 256-bit keys slightly differs.

The most popular and widely used application of Galois Field is in Cryptography. Since each byte of data are represented as a vector in a finite field, encryption and decryption using mathematical arithmetic is very straightforward and is easily manipulable

1.4 A.E.S Cipher

Like DES, AES is a symmetric block cipher. This means that it uses the same key for both encryption and decryption. However, AES is quite different from DES in a number of ways. The algorithm Rijndael allows for a variety of block and key sizes and not just the 64 and 56 bits of DES' block and key size. The block and key can in fact be chosen independently from 128, 160, 192, 224, 256 bits and need not be the same. However, the AES standard states that the algorithm can only accept a block size of 128 bits and a choice of three keys - 128, 192, 256 bits. Depending on which version is used, the name of the standard is modified to AES-128, AES-192 or AES- 256 respectively. As well as these differences AES differs from DES in that it is not a feistel structure. Recall that in a feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. In this case the entire data block is processed in parallel during each round using substitutions and permutations. A number of AES parameters depend on the key length. For example, if the key size used is 128 then the number of rounds is 10 whereas it is 12 and 14 for 192 and 256 bits respectively. At present the most common key size likely to be used is the 128 bit key. This description of the AES algorithm therefore describes this particular implementation.

Rijndael was designed to have the following characteristics:

• Resistance against all known attacks.

• Speed and code compactness on a wide range of platforms.

• Design Simplicity.

The overall structure of AES can be seen in the figure below. The input is a single 128 bit block both for decryption and encryption and is known as the in matrix. This block is copied into a state array which is modified at each stage of the algorithm and then copied to an output matrix. Both the plaintext and key are depicted as a 128 bit square matrix of bytes. This key is then expanded into an array of key schedule words (the w matrix). It must be

noted that the ordering of bytes within the in matrix is by column. The same applies to the w matrix.

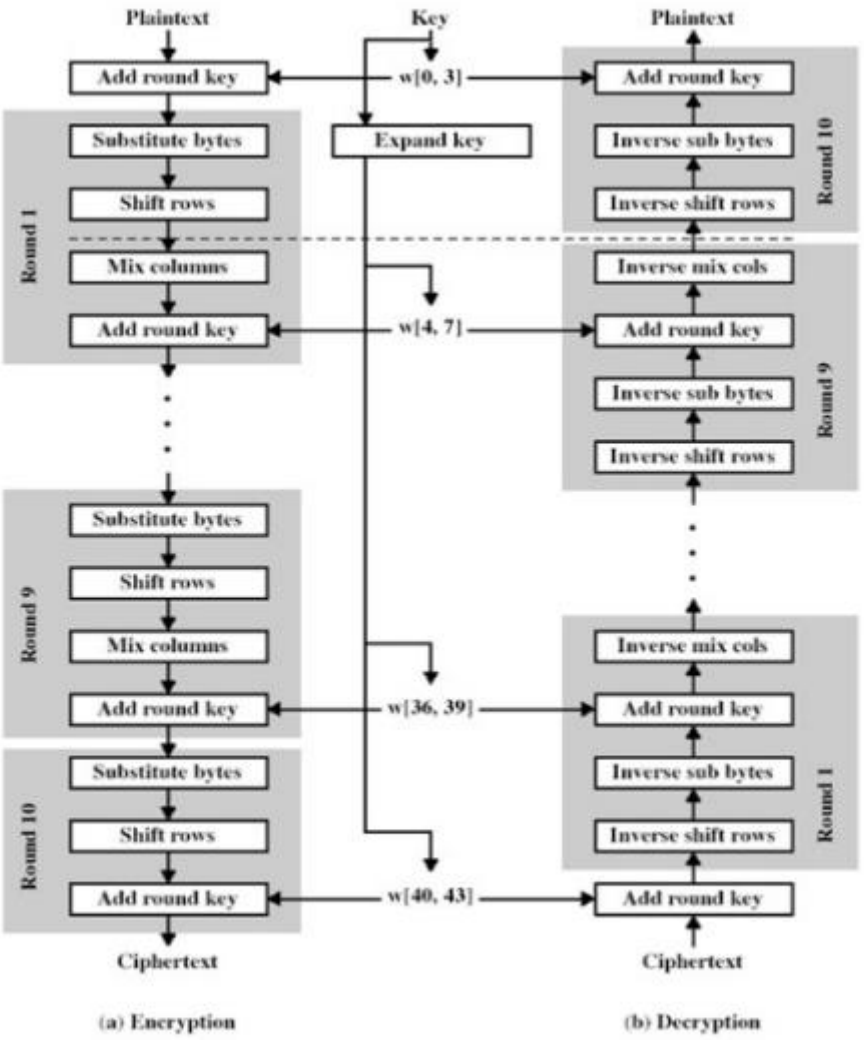

Figure 3 Structure of AES Cipher

The algorithm begins with an Add round key stage followed by 9 rounds of four stages and a tenth round of three stages. This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of it's counterpart in the encryption algorithm. The four stages are as follows:

1. Substitute bytes

2. Shift rows

3. Mix Columns

4. Add Round Key

The tenth round simply leaves out the Mix Columns stage. The first nine rounds of the decryption algorithm consist of the following:

1. Inverse Shift rows

2. Inverse Substitute bytes

3. Inverse Add Round Key

4. Inverse Mix Columns

Again, the tenth round simply leaves out the Inverse Mix Columns stage. Each of these stages will now be considered in more detail.

1.4.1 Substitute Bytes

This stage (known as SubBytes) is simply a table lookup using a 16×16 matrix of byte values called an s-box. This matrix consists of all the possible combinations of an 8 bit sequence (2 8 = 16 × 16 = 256). However, the s-box is not just a random permutation of these values and there is a well defined method for creating the s-box tables. The designers of Rijndael showed how this was done unlike the s-boxes in DES for which no rationale was given. We will not be too concerned here how the s-boxes are made up and can simply take them as table lookups. Again the matrix that gets operated upon throughout the encryption is known as state. We will be concerned with how this matrix is effected in each round. For this particular round each byte is mapped into a new byte in the following way: the leftmost nibble of the byte is used to specify a particular row of the s-box and the rightmost nibble specifies a column. For example, the byte {95} (curly brackets represent hex values in FIPS PUB 197) selects row 9 column 5 which turns out to contain the value {2A}. This is then used to update the state matrix.

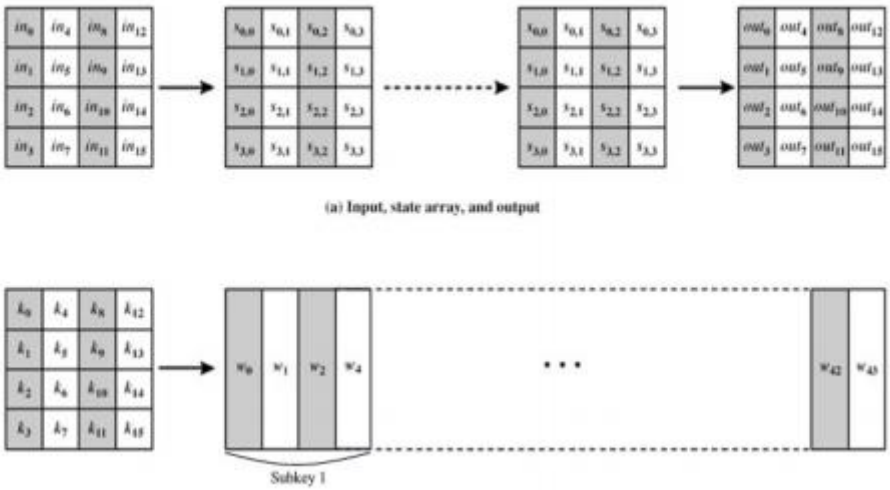

Figure 4 Matrix for Substituting bytes

1.4.2 Shift Row Transformation

This stage (known as ShiftRows) is shown in figure. This is a simple permutation an nothing more. It works as follow:

- The first row of state is not altered.
- The second row is shifted 1 bytes to the left in a circular manner.
- The third row is shifted 2 bytes to the left in a circular manner.
- The fourth row is shifted 3 bytes to the left in a circular manner.

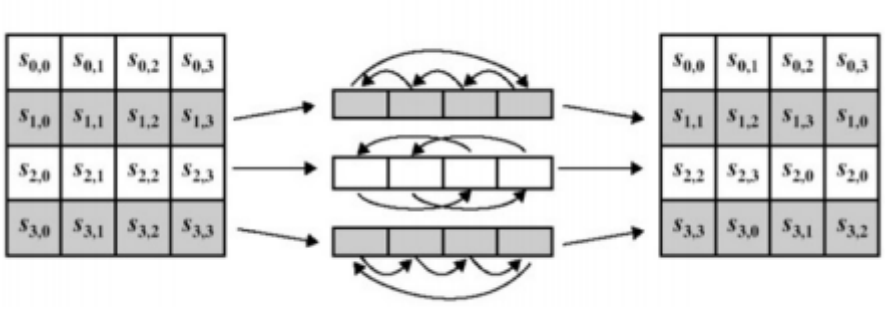

Figure 5 Shift Row Transformation

The Inverse Shift Rows transformation (known as InvShiftRows) performs these circular shifts in the opposite direction for each of the last three rows (the first row was unaltered to

begin with). This operation may not appear to do much but if you think about how the bytes are ordered within state then it can be seen to have far more of an impact. Remember that state is treated as an array of four byte columns, i.e. the first column actually represents bytes 1, 2, 3 and 4. A one byte shift is therefore a linear distance of four bytes. The transformation also ensures that the four bytes of one column are spread out to four different columns.

1.4.3 Mix Column Transformation

This stage (known as MixColumn) is basically a substitution but it makes use of arithmetic of GF(2^8). Each column is operated on individually. Each byte of a column is mapped into a new value that is a function of all four bytes in the column. The transformation can be determined by the following matrix multiplication on state.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Each element of the product matrix is the sum of products of elements of one row and one column. In this case the individual additions and multiplications are performed in GF(28). The MixColumns transformation of a single column j ($0 \leq j \leq 3$) of state can be expressed as:

$$\begin{aligned} s'_{0,j} &= (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j}) \\ s'_{3,j} &= (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j}) \end{aligned}$$

where • denotes multiplication over the finite field GF(2^8).

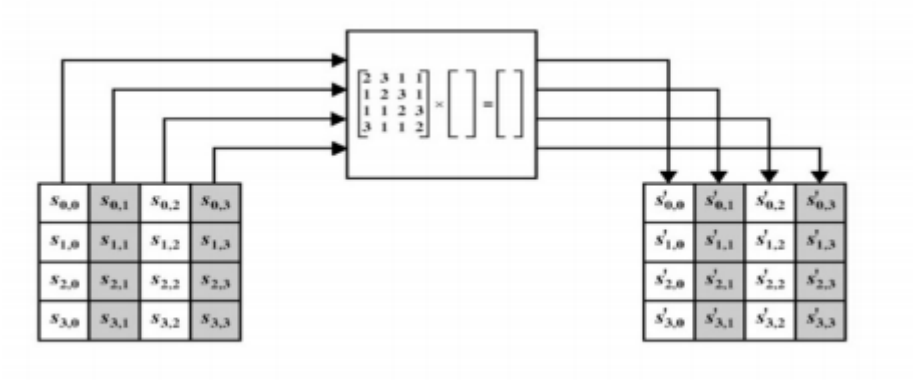

Figure 6 Mix Columns Transformation

1.4.4 Add Round Key Transformation

In this stage (known as AddRoundKey) the 128 bits of state are bitwise XORed with the 128 bits of the round key. The operation is viewed as a column wise operation between the 4 bytes of a state column and one word of the round key. This transformation is as simple as possible which helps in efficiency but it also effects every bit of state.

In each round of A.E.S there exist four sub processes. The first round is in the figure depicted below

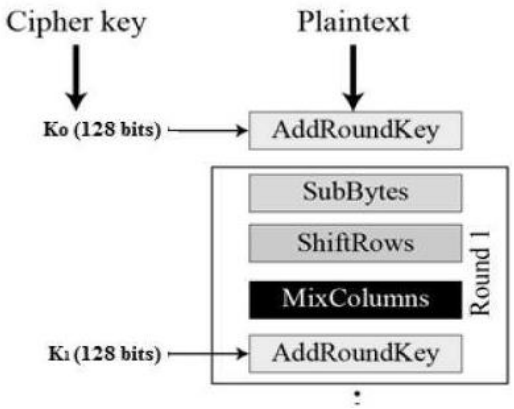

Figure 7 Add Round Key Transformation

The process of Mix Columns is used in the new approach which we are discussing in this paper. The Mix Columns step includes conversion of the four bytes using a special function which inputs the first four bytes in the first column and gives four new columns which

replaces the old four columns. The final output is another new matrix which contains new 16 bits. The transformation is as follows.

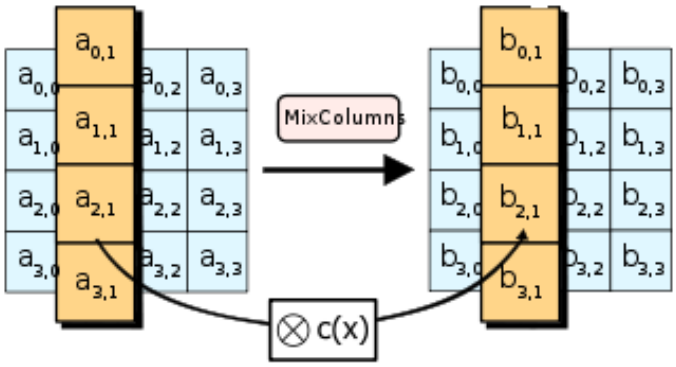

Figure 8 Mix Columns Transformation

The selection process for this new symmetric key algorithm was fully open to public scrutiny and comment; this ensured a thorough, transparent analysis of the designs submitted. NIST specified the new advanced encryption standard algorithm must be a block cipher capable of handling 128 bit blocks, using keys sized at 128, 192, and 256 bits; other criteria for being chosen as the next advanced encryption standard algorithm included:

- **Security:** Competing algorithms were to be judged on their ability to resist attack, as compared to other submitted ciphers, though security strength was to be considered the most important factor in the competition.
- **Cost:** Intended to be released under a global, nonexclusive and royalty-free basis, the candidate algorithms were to be evaluated on computational and memory efficiency.
- **Implementation:** Algorithm and implementation characteristics to be evaluated included the flexibility of the algorithm; suitability of the algorithm to be implemented in hardware or software; and overall, relative simplicity of implementation.

1.5 DLL's

Dynamic Link Libraries (DLL)s are like EXEs but they are not directly executable. They are similar to .so files in Linux/Unix. That is to say, DLLs are MS's implementation of shared libraries. DLLs are so much like an EXE that the file format itself is the same.

A DLL contains functions, classes, variables, UIs and resources (such as icons, images, files,...) that an EXE, or other DLL uses.

Types of libraries:

On virtually all operating systems, there are 2 types of libraries. Static libraries and dynamic libraries. In windows the file extensions are as follows: Static libraries (.lib) and dynamic libraries (.dll). The main difference is that static libraries are linked to the executable at compile time; whereas dynamic linked libraries are not linked until run-time.

1.5.1 Creating and using a DLL

To create a dynamic link library (DLL) project

1. On the menu bar, choose File**,** New**,** Project.
2. In the left pane of the New Project dialog box, expand Installed**,** Templates**,** Visual C++**,** and then select Win32.
3. In the center pane, select Win32 Console Application.
4. Specify a name for the project—for example, MathLibrary—in the Name box. Specify a name for the solution—for example, MathLibraryAndClient—in the Solution name box. Choose the OK button.
5. On the Overview page of the Win32 Application Wizard dialog box, choose the Next button.
6. On the Application Settings page, under Application type, select DLL.
7. Choose the Finish button to create the project.

1.5.2 Loading a DLL:

A program loads a DLL at startup, via the Win32 API LoadLibrary, or when it is a dependency of another DLL. A program uses the GetProcAddress to load a function or LoadResource to load a resource.

To add a class to the dynamic link library

1. To create a header file for a new class, on the menu bar, choose Project**,** Add New Item. In the Add New Item dialog box, in the left pane, select Visual C++. In the center pane, select Header File (.h). Specify a name for the header file—for example**,** MathLibrary.h—and then choose the Add button. A blank header file is displayed.

2. Replace the contents of the header file with this code:

```
#pragma once

#ifdef MATHLIBRARY_EXPORTS
#define MATHLIBRARY_API __declspec(dllexport)
#else
#define MATHLIBRARY_API __declspec(dllimport)
#endif

namespace MathLibrary
{
    // This class is exported from the MathLibrary.dll
    class Functions
    {
    public:
        // Returns a + b
        static MATHLIBRARY_API double Add(double a, double b);

        // Returns a * b
        static MATHLIBRARY_API double Multiply(double a, double b);

        // Returns a + (a * b)
        static MATHLIBRARY_API double AddMultiply(double a, double b);
    };
}
```

3. In the MathLibrary project in Solution Explorer, in the SourceFiles folder open MathLibrary.cpp

4. Implement the members of the Functions class in the source file. Replace the contents of the MathLibrary.cpp file with the following code:

```
// MathLibrary.cpp : Defines the exported functions for the DLL
application.
// Compile by using: cl /EHsc /DMATHLIBRARY_EXPORTS /LD MathLibrary.cpp

#include "stdafx.h"
#include "MathLibrary.h"

namespace MathLibrary
{
    double Functions::Add(double a, double b)
    {
        return a + b;
    }

    double Functions::Multiply(double a, double b)
    {
        return a * b;
    }

    double Functions::AddMultiply(double a, double b)
    {
```

```
        return a + (a * b);
    }
}
```

5. To verify that everything is working so far, compile the dynamic link library by choosing Build, Build Solution on the menu bar. The output should look something like this:

Output

```
1>------ Build started: Project: MathLibrary, Configuration: Debug Win32 --
----
1>  stdafx.cpp
1>  dllmain.cpp
1>  MathLibrary.cpp
1>     Creating library c:\users\username\documents\visual studio
2015\Projects\MathLibraryAndClient\Debug\MathLibrary.lib and object
c:\users\username\documents\visual studio
2015\Projects\MathLibraryAndClient\Debug\MathLibrary.exp
1>  MathLibrary.vcxproj -> c:\users\username\documents\visual studio
2015\Projects\MathLibraryAndClient\Debug\MathLibrary.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

CHAPTER – 2

LITERATURE SURVEY

1) AUTHORS: Prof. Subrata Das, Prof. Joy Prakash Sain

According to the author, he gave the justification of the use of third base. A complete architecture, design and implementation of 2 bit ALU slice were discussed. A new type of transmission functions theory was reported and with that theory the author suggests that this theory can explain all the CMOS ternary circuits. In this a mixed binary-ternary number system and its application in elliptic curve cryptosystem were discussed. Rotation Symmetric Boolean function has beckoned the interest of theoretician as well as practitioners in the field of cryptography. The author generalizes the counting results about *RSBFs* to the rotation symmetric Boolean polynomial over GF (p) where p is prime.

2) AUTHORS: Prof. Partha Sarathi Dasgupta, Prof. Samar Sensarma

It has been shown that Base-3 number system is nearly optimal for computation. In this paper we have studied some existing logical operation on ternary number system. They have also discussed some of the existing arithmetic operations using ternary number system. Some new algorithms for arithmetic operations have also been proposed, and shown to be quite efficient in time complexity.

3) AUTHORS: Prof. Erkay Savas, Prof. Cetin Kaya Koc

Cryptography is one of the most prominent application areas of the finite field arithmetic. Almost all public-key cryptographic algorithms including the recent algorithms such as elliptic curve and pairing-based cryptography rely heavily on finite field arithmetic, which needs to be performed efficiently to meet the execution speed and design space constraints. These objectives constitute massive challenges that necessitate interdisciplinary research efforts that will render the best algorithms, architectures, implementations, and design practices. They presented different architectures, methods and techniques for fast execution of cryptographic operations as well as high utilization of resources in the realization of cryptographic algorithms. While it is difficult to have a complete coverage of all related

work, they reflected the current trends and important implementation issues of finite field arithmetic in the context of cryptography.

4) AUTHORS: Prof. Desoky, Prof. Ashikhmin

Cryptography software system (CSS) is a set of tools to simulate and analyze a number of cryptography algorithms. It is written using Microsoft C# programming language and has a user friendly GUI. Arithmetic operations for encryption and decryption are in GF(2^8) and the analysis provides the user with the basic statistics of data before and after the application of the selected cryptography algorithm. Along with the implementation of five cryptography algorithms (affine, Vigenere, linear-feedback-shift-register, one-time-pad, and weighted sum), CSS is built modularly and the ability to add more algorithms is a definite advantage.

5) AUTHORS: Prof. Shabbir Hassan, Prof. Ubaidullah Bokhari

Nowadays fast computing and lightweight cryptography play a crucial role in the field of cryptography. Whenever we concern about the cryptography, the aspect of discrete mathematics can't be omitted. Security in cryptography is completely depends upon the key and the computations. Generation of key is nothing but the implementation of graph theory, discrete logarithms, linear & abstract algebra etc. These branches of modern mathematics has been playing a great role for the implementation of algorithms such as elliptic curve cryptography, stream cipher, block cipher, wireless sensor network etc. Their paper is written to literally represent the key concept of such branches of modern mathematics and their implementations in the field of cryptography. Their paper mainly deals the applications of Galois Field, primitive polynomials, primitive polynomials over Galois Field, Number theoretic functions, Congruence Calculus or modular arithmetic's, Residue Class Rings and Prime Fields. Their paper focus on these key topics to develop a mathematical tool, that are needed for the design and security analysis of a cryptosystems.

CHAPTER-3

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS:

- Processor : Pentium –III
- Speed : 1.1 Ghz
- RAM : 256 MB(minimum)
- Hard Disk : 20 GB
- Floppy Drive : 1.44 MB
- Key Board : Standard Windows Keyboard
- Mouse : Two or Three Button Mouse
- Monitor : SVGA

SOFTWARE REQUIREMENTS:

- Operating system : Windows XP
- Coding Language : .NET (C#)
- Software : Visual Studio

CHAPTER-4

PROJECTED FRAMEWORK

Before going into the projected framework, let us discuss about the problem which is raised with the existing system.

4.1 Disadvantage with the existing system:

The existing system consists of steps involving conversion of message to decimal, decimal to binary format, binary to ternary and then ternary to polynomial form. The disadvantage with this existing system is that it involves binary format which represents numbers in much longer format than any other number system. For instance, the decimal 99 in binary format is 1100011 which is a longer format and operations on such longer format includes complex algorithms which could even effect the run time complexity of the program. The average run time complexity for a large decimal like 99999 could result in $O(n^2)$ which effects the complexity.

Moreover the existence of decimal includes conversion of decimal to binary format to store in the computer as no computer stores decimal values. The decimal format lacks the prime factor 3 in it whereas the balanced ternary is definitely good enough, as it involves the cycle as follows. Let us consider the three digits 0,1,2 which in balanced ternary is 0,1,-1 are represented with 0,+,-. The counting follows as cycle as follows

+, +-, +0, ++, +--, +-0, +-+, +-0, ++-, ++0, ++

So, balanced ternary number system is completely fine with respect to storing the values in the computer. But storing of decimal values must involve conversion into binary formats which increases another step. So eliminating decimal step not only increases the precision of the system but also helps in encrypting the data efficiently. So, there must be an approach which can eliminate the complexity and thereby providing efficient method to encrypt the data.

4.2 Proposed Framework :

We consider the ASCII system in place of both decimal and binary systems. Because ASCII system contains 128 characters only whereas decimal number system has more than 120,000

characters in it. Moreover the ASCII systems need not to be converted to binary format to encrypt the data. It can be directly converted to ternary system without any conversion into binary number system. The new system proposed is as follows

Initially the characters of the message which is to be sent to the receiver are converted into its subsequent ASCII formats and the each ASCII subsequent value is converted into ternary number system. The subsequent ternary number for each character is then converted into its equivalent polynomial form and when all the characters polynomial forms are obtained the concept of Mix Columns in AES algorithm is used to combine all the polynomials by the operation of multiplication which involves addition algorithm also.

The Mix Columns of AES algorithm involves combining the four bytes of each column of the state using a linear transformation. As stated previously, multiplication with 1 is no change, multiplication with 2 means shifting to the left and multiplication with 3 means shifting to the left and then performing XOR operation with initial unshifted value. After shifting all the values are combined to obtain the result.

Making it simpler, each and every column is considered as a polynomial with Galois field $GF(2^3)$ as they are ternary values with only base-3 and then is multiplied with fixed polynomial. This polynomial is shared with the receiver which is used by the receiver in the process of decryption. Then the coefficients are displayed which the encrypted data of the message is sent.

4.3 Proposed Algorithm:

The below algorithm is used in encryption of a message by using galois field of degree 3^n and mix columns in Advanced Encryption Standard (A.E.S).

As an instance consider the message “Hello” is to be encrypted and is to be sent to the receiver and the steps involved are as follows

Step-1

The string that is to be encrypted is divided into characters and stored as {H, e, l, l, o}

Step-2

The characters of the message {H, e, l, l, o} are converted into their equivalent ASCII values and stored.

The ASCII values for the characters of the message are {72, 101, 108, 108, 111}.

Step-3

After obtaining the subsequent ASCII values of the characters of the message, the equivalent ternary values for each ASCII value is calculated and is stored.

The ternary values of the equivalent ASCII values are {02200, 10202, 11000, 11000, 11010}.

Step-4

Now, each ternary value is converted into its subsequent polynomial form as follows

Consider first ternary value i.e 02200, Lets say polynomial form for this value is p1(x).

$$\text{Now } p1(x) = (0)(x^0) + (0)(x^1) + (-1)(x^2) + (-1)(x^3) + (0)(x^4)$$

$$p1(x) = -x^2 - x^3$$

$$\text{Similarly, } p2(x) = (-1)(x^0) + (0)(x^1) + (-1)(x^2) + (0)(x^3) + (1)(x^4)$$

$$p2(x) = -1 - x^2 + x^4$$

$$\text{Similarly, } p3(x) = (0)(x^0) + (0)(x^1) + (0)(x^2) + (1)(x^3) + (1)(x^4)$$

$$p3(x) = x^3 + x^4$$

$$\text{Similarly, } p4(x) = p3(x)$$

$$\text{And, } p5(x) = (0)(x^0) + (1)(x^1) + (0)(x^2) + (1)(x^3) + (1)(x^4)$$

$$p5(x) = x^1 + x^3 + x^4$$

During conversion of ternary system into polynomial form we assume that $x\text{^}k^1$ is the coefficient of each polynomial form where the power of k varies for each term from 1 to k, where k is nothing but the number of characters in the message.

The Polynomial form can be represented as,

$$p1(x) + (x^5)\,p2(x) + (x^5)^2\,p3(x) + (x^5)^3\,p4(x) + (x^5)^4\,p5(x),$$

Here in x^5 , the power 5 represents the number of characters in the message.

Now, the polynomial form is as follows

$$p(x) = -x^2 - x^3 +[x^5(-1 - x^2 - x^4)] + [x^{10}(x^3 + x^4)] + [x^{15}(x^3 + x^4)] + [x^{20} (x^1+x^3+x^4)]$$

On solving we get,

$$p(x) = x^{24} + x^{23} + x^{21} + x^{19} + x^{18} + x^{14} + x^{13} – x^9 –x^7 –x^5 –x^3 – x^2 \text{ i.e}$$

$p(x) = (1) (x^{24}) + (1) (x^{23}) + (0) (x^{22}) + (1) (x^{21}) + (0) (x^{20}) + (1) (x^{19}) + (1) (x^{18}) + (0) (x^{17}) + (0) (x^{16}) + (0) (x^{15)} + (1) (x^{14}) + (1) (x^{13}) + (0) (x^{12}) + (0) (x^{11}) + (0) (x^{10}) + (-1) (x^9) + (0) (x^8) + (-1) (x^7) + (0) (x^6) + (-1) (x^5) + (0) (x^4) + (-1) (x^3) + (-1) (x^2) + (0) (x^1) + (0) (x^0)$

So, the coefficients are,

$$\{0, 0, -1, -1, 0, -1, 0, -1, 0, -1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1\}$$

which is the encrypted data which is to be sent to the receiver.

The receiver uses the common function which is used in encryption to decrypt the message.

These are the four steps involved in our proposed system which eliminates the usage of decimal and binary values and increases the efficiency of encryption by using ASCII system.

CHAPTER-5

SOURCE CODE

5.1 EncryptedAlgorithm.cs

```
using PolyLib;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace EncryptedDecryptedAlgo
{
   public partial class Form1 : Form
   {
      public string myString;
      public static int[] myAsCIIArray;
      public static string[] myTernaryArray;
      public Form1()
      {
         InitializeComponent();
      }
      private void btnEncrypt_Click(object sender, EventArgs e)
      {
         myString = txtText.Text;
         if (txtText.Text != "")
         {
            myAsCIIArray = new int[myString.Length];
            int i = 0;
```

```
        foreach (char mychar in myString)
        {
            int unicode = mychar;
            txtASCII.Text += unicode;
            myAsCIIArray[i] = unicode;
            i++;
        }
        string ter = ToTrenary(myAsCIIArray);
        txtTernary.Text = ter;
        ChangeToPolynomial(myTernaryArray);
    }
}

public void ChangeToPolynomial(string[] myTerArray)
{
    Polynomial[] polyArray = new Polynomial[myTerArray.Length];
    for (int i = 0; i < myTerArray.Length; i++)
    {
        double [] myDoubleArray=new double[myTerArray[i].Length];
        for (int j = 0; j < myTerArray[i].Length; j++)
        {
            if (myTerArray[i][j].ToString() == "2")
            {
                myDoubleArray[j] = -1;
            }
            else
            {
                myDoubleArray[j] = double.Parse(myTerArray[i][j].ToString());
            }
        }
        polyArray[i] = new Polynomial(myDoubleArray);
    }
    Polynomial[] myFinalPolynomialParts = new Polynomial[myString.Length];
    for (int k = 0; k < myString.Length; k++)
```

```
    {

        if(k==0){
            myFinalPolynomialParts[k]=polyArray[k];
        }
        else{

        double[] dZeroSequence=new double[(myString.Length*k)+1];
        dZeroSequence[(myString.Length*k)] = 1;
        myFinalPolynomialParts[k] = polyArray[k] * (new Polynomial(dZeroSequence));
        }
    }
    Polynomial myFinalPolynomial=null;
    for (int l = 0; l < myFinalPolynomialParts.Length; l++)
    {
        if (myFinalPolynomial == null)
        {
            myFinalPolynomial = myFinalPolynomialParts[l];
        }
        else
        {
            myFinalPolynomial += myFinalPolynomialParts[l];
        }
    }
   txtPolyEncryptedFormat.Text = myFinalPolynomial.ToString();
}
public string ToTrenary(int[] myAsCIIArray)
{
    StringBuilder Sb = new StringBuilder();
    myTernaryArray = new string[myAsCIIArray.Length];
    for (int i = 0; i < myAsCIIArray.Length; i++)
    {
        string myTerValues = "";
        while (myAsCIIArray[i] > 0)
```

```
            {
                if (myAsCIIArray[i] % 3 == 2)
                {
                    Sb.Insert(0, -1);
                }
                else
                {
                    Sb.Insert(0, myAsCIIArray[i] % 3);
                }
                myTerValues += (myAsCIIArray[i] % 3).ToString();
                myAsCIIArray[i] /= 3;
            }
            myTernaryArray[i] = myTerValues;
        }
        return Sb.ToString();
    }
    private void button1_Click(object sender, EventArgs e)
    {
        clear();
    }
    void clear()
    {
        txtText.Clear();
        txtASCII.Clear();
        txtTernary.Clear();
        txtPolyEncryptedFormat.Clear();
        txtDecrypted.Clear();
    }
    private void button2_Click(object sender, EventArgs e)
    {
        txtDecrypted.Text = myString;
    }
  }
}
```

5.2 Polynomial.dll

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.Collections;

namespace PolyLib
{
   public class Polynomial
   {
      #region Fields
      /// <summary>
      /// Coefficients a_0,...,a_n of a polynomial p, such that
      /// p(x) = a_0 + a_1*x + a_2*x^2 + ... + a_n*x^n.
      /// </summary>
      public Complex[] Coefficients;
      #endregion
      #region constructors
      /// <summary>
      /// Inits zero polynomial p = 0.
      /// </summary>
      public Polynomial()
      {
         Coefficients = new Complex[1];
         Coefficients[0] = Complex.Zero;
      }
      /// <summary>
      /// Inits polynomial from given complex coefficient array.
      /// </summary>
      /// <param name="coeffs"></param>
      public Polynomial(params Complex[] coeffs)
      {
```

```
    if (coeffs == null || coeffs.Length < 1)
    {
        Coefficients = new Complex[1];
        Coefficients[0] = Complex.Zero;
    }
    else
    {
        Coefficients = (Complex[])coeffs.Clone();
    }
}

/// <summary>
/// Inits polynomial from given real coefficient array.
/// </summary>
/// <param name="coeffs"></param>
public Polynomial(params double[] coeffs)
{
    if (coeffs == null || coeffs.Length < 1)
    {
        Coefficients = new Complex[1];
        Coefficients[0] = Complex.Zero;
    }
    else
    {
        Coefficients = new Complex[coeffs.Length];
        for (int i = 0; i < coeffs.Length; i++)
            Coefficients[i] = new Complex(coeffs[i]);
    }
}
/// <summary>
/// Inits constant polynomial.
/// </summary>
/// <param name="coeffs"></param>
public Polynomial(Complex c)
```

```
{
    Coefficients = new Complex[1];
    if (c == null)
        Coefficients[0] = Complex.Zero;
    else
        Coefficients[0] = c;
}

/// <summary>
/// Inits constant polynomial.
/// </summary>
/// <param name="coeffs"></param>
public Polynomial(double c)
{
    Coefficients = new Complex[1];

    Coefficients[0] = new Complex(c);
}
/// <summary>
/// Inits polynomial from string like "2x^2 + 4x + (2+2i)"
/// </summary>
/// <param name="p"></param>
public Polynomial(string p)
{
    throw new NotImplementedException();
}
#endregion
#region dynamics
/// <summary>
/// Computes value of the differentiated polynomial at x.
/// </summary>
/// <param name="p"></param>
/// <param name="x"></param>
/// <returns></returns>
```

```
public int Degree
{
    get
    {
        return Coefficients.Length - 1;
    }
}
/// <summary>
/// Checks if given polynomial is zero.
/// </summary>
/// <returns></returns>
public bool IsZero()
{
    for (int i = 0; i < Coefficients.Length; i++)
        if (Coefficients[i] != 0) return false;
    return true;
}
/// <summary>
/// Evaluates polynomial by using the horner scheme.
/// </summary>
/// <param name="x"></param>
/// <returns></returns>
public Complex Evaluate(Complex x)
{
    Complex buf = Coefficients[Degree];
    for (int i = Degree - 1; i >= 0; i--)
    {
        buf = Coefficients[i] + x * buf;
    }
    return buf;
}
/// <summary>
/// Normalizes the polynomial, e.i. divides each coefficient by the
/// coefficient of a_n the greatest term if a_n != 1.
```

```
/// </summary>
public void Normalize()
{
   this.Clean();

   if (Coefficients[Degree] != Complex.One)
      for (int k = 0; k <= Degree; k++)
         Coefficients[k] /= Coefficients[Degree];
}
/// <summary>
/// Removes unnecessary zero terms.
/// </summary>
public void Clean()
{
   int i;
   for (i = Degree; i >= 0 && Coefficients[i] == 0; i--) ;
   Complex[] coeffs = new Complex[i + 1];

   for (int k = 0; k <= i; k++)
      coeffs[k] = Coefficients[k];

   Coefficients = (Complex[])coeffs.Clone();
}

/// <summary>
/// Factorizes polynomial to its linear factors.
/// </summary>
/// <returns></returns>
public FactorizedPolynomial Factorize()
{
   // this is to be returned
   FactorizedPolynomial p = new FactorizedPolynomial();
   // cannot factorize polynomial of degree 0 or 1
   if (this.Degree <= 1)
```

```
        {
            p.Factor = new Polynomial[] { this };
            p.Power = new int[] { 1 };

            return p;
        }

        Complex[] roots = Roots(this);

        //ArrayList rootlist = new ArrayList();
        //foreach (Complex z in roots) rootlist.Add(z);
        //roots = null; // don't need you anymore
        //rootlist.Sort();
        //// number of different roots
        //int num = 1; // ... at least one
        //// ...or more?
        //for (int i = 1; i < rootlist.Count; i++)
        //    if (rootlist[i] != rootlist[i - 1]) num++;

        Polynomial[] factor = new Polynomial[roots.Length];
        int[] power = new int[roots.Length];

        //factor[0] = new Polynomial( new Complex[]{ -(Complex)rootlist[0] *
Coefficients[Degree],
        //    Coefficients[Degree] } );
        //power[0] = 1;

        //num = 1;
        //len = 0;
        //for (int i = 1; i < rootlist.Count; i++)
        //{
        //    len++;
        //    if (rootlist[i] != rootlist[i - 1])
        //    {
```

```
            //              factor[num] = new Polynomial(new Complex[] { -(Complex)rootlist[i], Complex.One });
            //      power[num] = len;
            //      num++;
            //      len = 0;
            //  }
            //}
            power[0] = 1;
            factor[0] = new Polynomial(new Complex[] { -Coefficients[Degree] * (Complex)roots[0], Coefficients[Degree] });

            for (int i = 1; i < roots.Length; i++)
            {
                power[i] = 1;
                factor[i] = new Polynomial(new Complex[] { -(Complex)roots[i], Complex.One });
            }

            p.Factor = factor;
            p.Power = power;

            return p;
        }

        /// <summary>
        /// Computes the roots of polynomial via Weierstrass iteration.
        /// </summary>
        /// <returns></returns>
        public Complex[] Roots()
        {
            double tolerance = 1e-12;
            int max_iterations = 30;

            Polynomial q = Normalize(this);
            //Polynomial q = p;
```

```
        Complex[] z = new Complex[q.Degree]; // approx. for roots
        Complex[] w = new Complex[q.Degree]; // Weierstraß corrections

        // init z
        for (int k = 0; k < q.Degree; k++)
            //z[k] = (new Complex(.4, .9)) ^ k;
            z[k] = Complex.Exp(2 * Math.PI * Complex.I * k / q.Degree);


        for (int iter = 0; iter < max_iterations
            && MaxValue(q, z) > tolerance; iter++)
            for (int i = 0; i < 10; i++)
            {
                for (int k = 0; k < q.Degree; k++)
                    w[k] = q.Evaluate(z[k]) / WeierNull(z, k);

                for (int k = 0; k < q.Degree; k++)
                    z[k] -= w[k];
            }

        // clean...
        for (int k = 0; k < q.Degree; k++)
        {
            z[k].Re = Math.Round(z[k].Re, 12);
            z[k].Im = Math.Round(z[k].Im, 12);
        }

        return z;
    }

    /// <summary>
    /// Expands factorized polynomial p_1(x)^(k_1)*...*p_r(x)^(k_r) to its normal form a_0
+ a_1 x + ... + a_n x^n.
```

```
/// </summary>
/// <param name="p"></param>
/// <returns></returns>
public static Polynomial Expand(FactorizedPolynomial p)
{
   Polynomial q = new Polynomial(new Complex[] { Complex.One });

   for (int i = 0; i < p.Factor.Length; i++)
   {
      for (int j = 0; j < p.Power[i]; j++)
         q *= p.Factor[i];

      q.Clean();
   }

   // clean...
   for (int k = 0; k <= q.Degree; k++)
   {
      q.Coefficients[k].Re = Math.Round(q.Coefficients[k].Re, 12);
      q.Coefficients[k].Im = Math.Round(q.Coefficients[k].Im, 12);
   }

   return q;
}

/// <summary>
/// Evaluates factorized polynomial p at point x.
/// </summary>
/// <param name="p"></param>
/// <returns></returns>
public static Complex Evaluate(FactorizedPolynomial p, Complex x)
{
   Complex z = Complex.One;
```

```
    for (int i = 0; i < p.Factor.Length; i++)
    {
        z *= Complex.Pow(p.Factor[i].Evaluate(x), p.Power[i]);
    }

    return z;
}

/// <summary>
/// Removes unncessary leading zeros.
/// </summary>
/// <param name="p"></param>
/// <returns></returns>
public static Polynomial Clean(Polynomial p)
{
    int i;

    for (i = p.Degree; i >= 0 && p.Coefficients[i] == 0; i--) ;
    Complex[] coeffs = new Complex[i + 1];

    for (int k = 0; k <= i; k++)
        coeffs[k] = p.Coefficients[k];

    return new Polynomial(coeffs);
}

/// <summary>
/// Normalizes the polynomial, e.i. divides each coefficient by the
/// coefficient of a_n the greatest term if a_n != 1.
/// </summary>
public static Polynomial Normalize(Polynomial p)
{
    Polynomial q = Clean(p);
    if (q.Coefficients[q.Degree] != Complex.One)
```

```
        for (int k = 0; k <= q.Degree; k++)
            q.Coefficients[k] /= q.Coefficients[q.Degree];
    return q;
}

/// <summary>
/// Computes the roots of polynomial p via Weierstrass iteration.
/// </summary>
/// <param name="p"></param>
/// <returns></returns>
public static Complex[] Roots(Polynomial p)
{
    double tolerance = 1e-12;
    int max_iterations = 30;

    Polynomial q = Normalize(p);
    //Polynomial q = p;

    Complex[] z = new Complex[q.Degree]; // approx. for roots
    Complex[] w = new Complex[q.Degree]; // Weierstraß corrections

    // init z
    for (int k = 0; k < q.Degree; k++)
        //z[k] = (new Complex(.4, .9)) ^ k;
        z[k] = Complex.Exp(2 * Math.PI * Complex.I * k / q.Degree);
    for (int iter = 0; iter < max_iterations
        && MaxValue(q, z) > tolerance; iter++)
        for (int i = 0; i < 10; i++)
        {
            for (int k = 0; k < q.Degree; k++)
                w[k] = q.Evaluate(z[k]) / WeierNull(z, k);

            for (int k = 0; k < q.Degree; k++)
                z[k] -= w[k];
```

```
            }

        // clean...
        for (int k = 0; k < q.Degree; k++)
        {
            z[k].Re = Math.Round(z[k].Re, 12);
            z[k].Im = Math.Round(z[k].Im, 12);
        }

        return z;
    }

    /// <param name="tolerance">Computation precision; e.g. 1e-12 denotes 12 exact digits.</param>
    /// <param name="max_iterations">Maximum number of iterations; this value is used to bound

    public static Complex[] Roots(Polynomial p, double tolerance, int max_iterations)
    {
        Polynomial q = Normalize(p);

        Complex[] z = new Complex[q.Degree]; // approx. for roots
        Complex[] w = new Complex[q.Degree]; // Weierstraß corrections

        // init z
        for (int k = 0; k < q.Degree; k++)
            //z[k] = (new Complex(.4, .9)) ^ k;
            z[k] = Complex.Exp(2 * Math.PI * Complex.I * k / q.Degree);


        for (int iter = 0; iter < max_iterations
            && MaxValue(q, z) > tolerance; iter++)
            for (int i = 0; i < 10; i++)
            {
```

```
            for (int k = 0; k < q.Degree; k++)
                w[k] = q.Evaluate(z[k]) / WeierNull(z, k);

            for (int k = 0; k < q.Degree; k++)
                z[k] -= w[k];
        }

    // clean...
    for (int k = 0; k < q.Degree; k++)
    {
        z[k].Re = Math.Round(z[k].Re, 12);
        z[k].Im = Math.Round(z[k].Im, 12);
    }

    return z;
}


private static Complex WeierNull(Complex[] z, int k)
{
    if (k < 0 || k >= z.Length)
        throw new ArgumentOutOfRangeException();
   Complex buf = Complex.One;
   for (int j = 0; j < z.Length; j++)
            if (j != k) buf *= (z[k] - z[j])
    return buf;
}
/// <summary>
/// Differentiates given polynomial p.
public static Polynomial Integral(Polynomial p)
{
    Complex[] buf = new Complex[p.Degree + 2];
    buf[0] = Complex.Zero; // this value can be arbitrary, in fact
```

```
    for (int i = 1; i < buf.Length; i++)
        buf[i] = p.Coefficients[i - 1] / i;
    return new Polynomial(buf);
}

/// <summary>
/// Computes the monomial x^degree.
/// </summary>
/// <param name="degree"></param>
/// <returns></returns>
public static Polynomial Monomial(int degree)
{
    if (degree == 0) return new Polynomial(1);

    Complex[] coeffs = new Complex[degree + 1];

    for (int i = 0; i < degree; i++)
        coeffs[i] = Complex.Zero;

    coeffs[degree] = Complex.One;

    return new Polynomial(coeffs);
}


public static Polynomial operator +(Polynomial p, Polynomial q)
{

    int degree = Math.Max(p.Degree, q.Degree);

    Complex[] coeffs = new Complex[degree + 1];

    for (int i = 0; i <= degree; i++)
    {
```

```
        if (i > p.Degree) coeffs[i] = q.Coefficients[i];
        else if (i > q.Degree) coeffs[i] = p.Coefficients[i];
        else coeffs[i] = p.Coefficients[i] + q.Coefficients[i];
    }

    return new Polynomial(coeffs);
}

public static Polynomial operator -(Polynomial p, Polynomial q)
{
    return p + (-q);
}

public static Polynomial operator -(Polynomial p)
{
    Complex[] coeffs = new Complex[p.Degree + 1];

    for (int i = 0; i < coeffs.Length; i++)
        coeffs[i] = -p.Coefficients[i];

    return new Polynomial(coeffs);
}

public static Polynomial operator *(Complex d, Polynomial p)
{
    Complex[] coeffs = new Complex[p.Degree + 1];

    for (int i = 0; i < coeffs.Length; i++)
        coeffs[i] = d * p.Coefficients[i];

    return new Polynomial(coeffs);
}

public static Polynomial operator *(Polynomial p, Complex d)
```

```
{
    Complex[] coeffs = new Complex[p.Degree + 1];

    for (int i = 0; i < coeffs.Length; i++)
        coeffs[i] = d * p.Coefficients[i];

    return new Polynomial(coeffs);
}

public static Polynomial operator *(double d, Polynomial p)
{
    Complex[] coeffs = new Complex[p.Degree + 1];

    for (int i = 0; i < coeffs.Length; i++)
        coeffs[i] = d * p.Coefficients[i];

    return new Polynomial(coeffs);
}

public static Polynomial operator *(Polynomial p, double d)
{
    Complex[] coeffs = new Complex[p.Degree + 1];

    for (int i = 0; i < coeffs.Length; i++)
        coeffs[i] = d * p.Coefficients[i];

    return new Polynomial(coeffs);
}


public string ToString(string format)
{
    if (this.IsZero()) return "0";
    else
```

```
    {
        string s = "";

        for (int i = 0; i < Degree + 1; i++)
        {
            if (Coefficients[i] != Complex.Zero)
            {
                if (Coefficients[i] == Complex.I)
                    s += "i";
                else if (Coefficients[i] != Complex.One)
                {
                    if (Coefficients[i].IsReal() && Coefficients[i].Re > 0)
                        s += Coefficients[i].ToString(format);
                    else
                        s += "(" + Coefficients[i].ToString(format) + ")";

                }
                else if (/*Coefficients[i] == Complex.One && */i == 0)
                    s += 1;

                if (i == 1)
                    s += "x";
                else if (i > 1)
                    s += "x^" + i.ToString(format);
            }

            if (i < Degree && Coefficients[i + 1] != 0 && s.Length > 0)
                s += " + ";
        }

        return s;
    }
}
```

CHAPTER-6

OUTPUTS

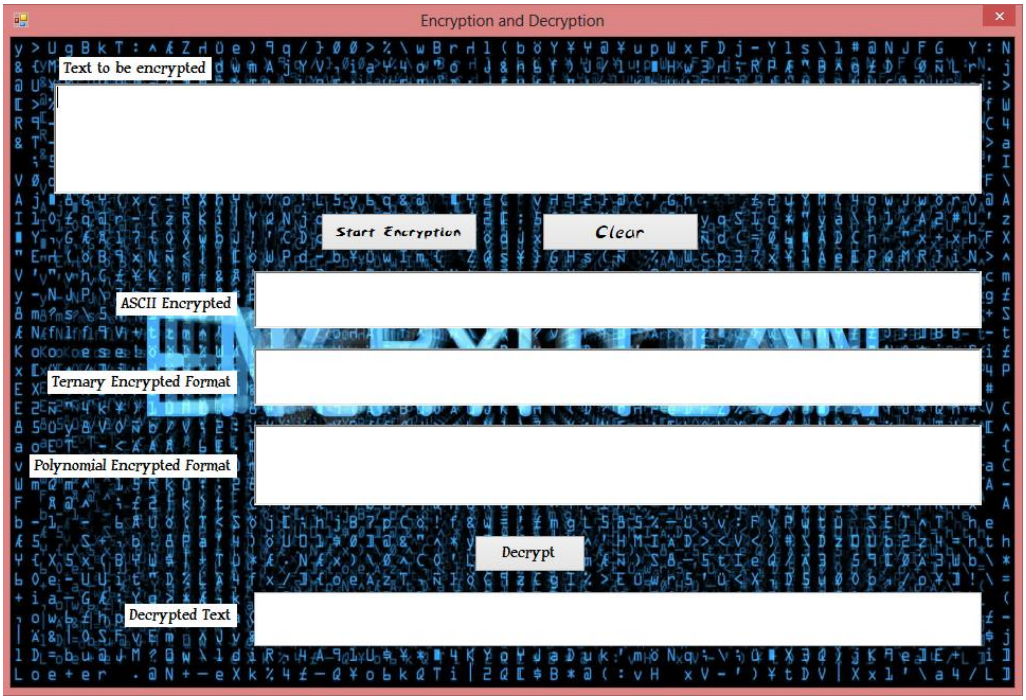

Figure 9 Encryption Interface

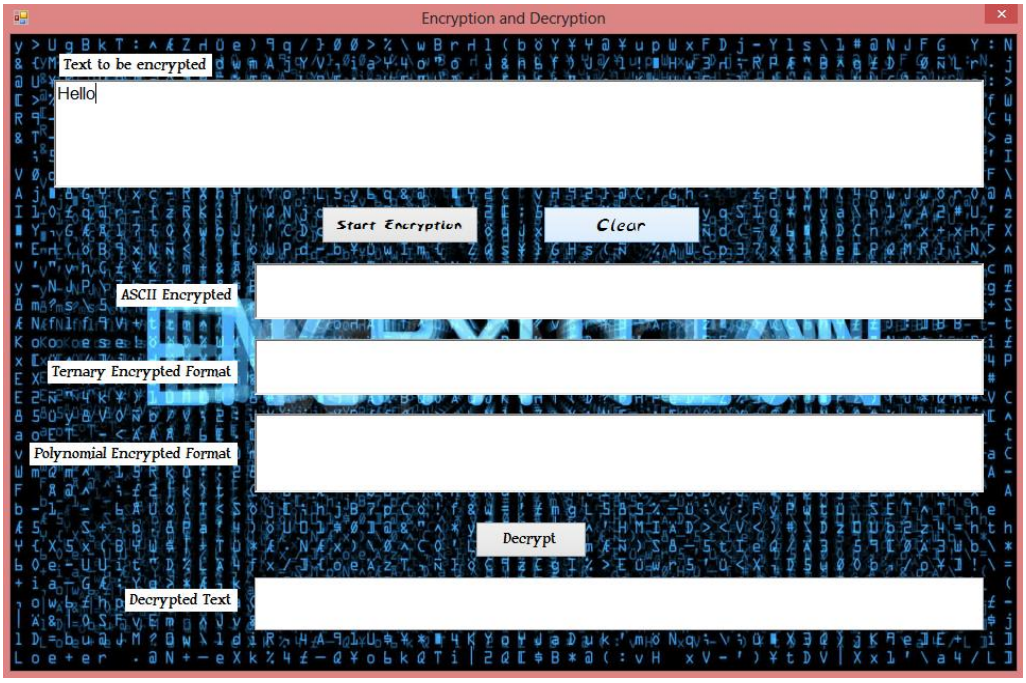

Figure 10 Entering Message that is to be encrypted

Figure 11 Conversion of Message to ASCII Format

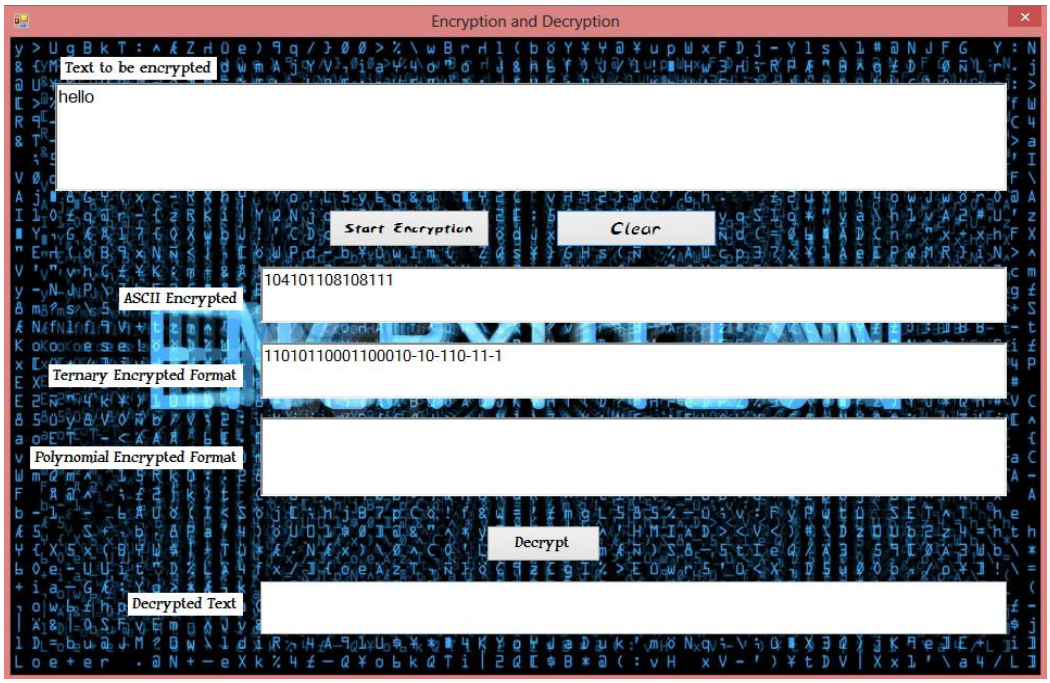

Figure 12 Conversion of ASCII Format to Ternary format

Figure 13 Encrypted ASCII, Ternary and Polynomial formats of Message by using Mix-Columns and Galois Field

Figure 14 Decryption of Encrypted Message

CHAPTER-7
CONCLUSION

In this project, we implemented a new algorithm for conversion of message into polynomial form which involves the usage of Mix columns algorithm in Advanced Encryption Standard (AES) and multiplication algorithm of ternary system. In this new algorithm, we eliminated the steps of decimal and binary systems by replacing them with ASCII system which is much more efficient than those two systems. As far as it is concerned, this is the first algorithm which involves only Mix columns of AES algorithm to encrypt the data. In terms of computational complexity this system produced better results with time complexity O(n) which is far better than the previous system which has time complexity of $O(2n^2) + O(n)$.

CHAPTER-8

REFERENCES

1. P Stanica , Rotation Symmetric Boolean Functions and Cryptographic Properties, ,Vol-156,n0.-10,May 2008.

2. J. Pieprzyk, Rotation-Symmetric Functions, Journal of Universal Computer Science,pp 20-31,Vol. 5, no.1 (1999).

3. Yuan Li, Results on Rotation Symmetric Polynomials Over GF(3),Elsevier, Science Direct.

4. Brian Hayes, Third Base, American scientist, Vol. 89, No.6, 2001, pp. 490-494

5. A Stakhov, Brousentsov's Ternary Principle, and Ternary Mirror-Symmetrical Arithmetic, The Computer Journal, Vol 45,N0. 2, 2002.

6. Vasundara Patel, K S Gurumurthy, Multi-valued Logic Addition and Multiplication in Galois Field, Control,and Telecommunication Technologies, 2009.

7. Subrata Das, Joy Prakash Sain, Parthasarathi Dasgupta, Algorithms for Ternary Number System, 2nd International Conference on Computer, Elsevier Science Direct,Procedia technology,Vol-4,pp- 278-285.

8. Subrata Das and Samar Sensarma, Arithmetic Algorithms for Ternary Number System,VDAT- 2012,Springer,Vol-7373,pp111-120,2012.

9. A.P.Dhande, V.T. Ingole, Design and Implementation of 2-Bit ternary ALU Slice, 3 rd International Conference SETIT, 2005, Tunisia.

10. X.W.Wu, CMOS Ternary Logic Circuits, Vol 137, Pt.G, No.1, Feb 1990.

11. A Stakhov, Brousentsov's Ternary Principle, Bergman's Number System and Ternary Mirror-Symmetrical Arithmetic, The Computer Journal, Vol 45, No. 2, 2002.

12. J. Adikari, V. S. Dimitrov, L. Imbert, Hybrid Binary-Ternary Number System for Elliptic Curve Crypto System, IEEE Transactions on Computers, Vol. 60, No. 2, Feb 2011.

13. P.C.Balla and A.Antoniou, Low Power dissipation of MOS Ternary logic Family, Vol. Sc-19, No.5, 1994.

14. S. Lin, Y-B Kim,F.Lombardi, CNTFET Based Design of Ternary Logic Gates and Arithmetic Circuits, IEEE Transactions of Nanotechnology, Vol.2, No.11, 2011.