**Crypto Library Project**

**Caesar Cipher:**

* **Functions used in enc , dec , cryptanalysis**

const string allAlphabet{ "ABCDEFGHIJKLMNOPQRSTUVWXYZ"}

* Constant string variable carries all alphabet characters from a to z

void textToUpper()

* Input : plaintext
* Convert plaintext to upper characters

allAlphabet.find()

* Find function search for character in string and returns index of it

int mod(int k, int n)

* To find character before shifting in alphabet
* Input : index of character , length of alphabet(26)
* Output : index of character before shifting

1. **Encryption**

string EncryptCaeser(string plainText, int keyValue)

* Input : plaintext , key
* Output : Cipher text

1. **Decryption**

string DecryptCaeser(string EncryptedText, int keyValue)

* Input : Cipher text , key
* Output : Plain text

1. **CryptAnalysis**

void bruteforceCaeser(string EncryptedText)

* Input : Cipher Text
* Print all possible keys with its own plaintext

int cryptanalysis(string EncryptedText, string plain\_txt)

* Input : cipher text , plaint text
* Output : key

**Mono Cipher :**

1. **Encryption**

String encrypt(string plaintext, string key)

* + Input : plain text , key
  + Output : cipher text

1. **Decryption**

String decrypt(string ciphertext, string key)

* + Input : cipher text , key
  + Output : plain text

1. **Analysis**

void AnalyseUsingCharFrequency(string ciphertext)

* + Input : cipher text
  + Output : display the result of frequency analysis on cipher text

**PlayFair :**

void create\_matrix()

* Input : key as string
* Create matrix of key

void encrypt(string plain\_txt)

* Input : plain text
* Print cipher text

void decrypt(string cipher\_txt)

* Input : Cipher Text
* Output : plain Text

**Rail Fence :**

plain\_txt.erase(remove(plain\_txt.begin(), plain\_txt.end(), ' '), plain\_txt.end())

* Remove spaces from string

transform(plain\_txt.begin(), plain\_txt.end(), plain\_txt.begin(), toupper)

* Convert string to upper case

ceil(int)

* returns the smallest possible integer value which is greater than or equal to the given argument.

1. **Encryption**

Encrypt\_RainFence(string plain\_txt, int key)

* Input : Plain text , key
* Output : Cipher text

1. **Decryption**

Decrypt\_RainFence(string cipher\_txt, int key)

* Input : Cipher text , key
* Output : Plain text

1. **CryptAnalysis**

int Analyse\_RainFence(string plain\_txt, string cipher\_txt)

* Input : Plain text , Cipher text
* Output : key steps

**Vigenere with Auto key :**

void textToUpper(string& text)

* Input : plaintext
* Convert plaintext to upper characters

int mod(int k, int n)

* To find character before shifting in alphabet
* Input : index of character , length of alphabet(26)
* Output : index of character before shifting

**(1)Encryption**

string Autokeystream(string PlainTEXT, string Key);

* Fill the difference in length between key and plain (key = key +plain[i])
* Input : plain text , key
* Output : key with same length of plain

string Encrypt(string plainText, string key)

* Input : plain text , key
* Output : Cipher Text

**(2)Decryption**

* string Decrypt(string cipherText, string key);
* Input : Cipher text , key
* Output : Plain text

**(3)CryptAnalysis**

* string Analyse(string plainText, string cipherText)
* Input : Plain text , Cipher text
* Output : key string

**Vigenere with Repeating key :**

void textToUpper(string& text)

* Input : plaintext
* Convert plaintext to upper characters

int mod(int k, int n)

* To find character before shifting in alphabet
* Input : index of character , length of alphabet(26)
* Output : index of character before shifting

**(1)Encryption**

string Repeatingkeystream(string PlainTEXT, string Key)

* Fill the difference in length between key and plain (key = key +key[i])
* Input : plain text , key
* Output : key with same length of plain

string Encrypt(string plainText, string key)

* Input : plain text , key
* Output : Cipher Text

**(2)Decryption**

string Decrypt(string cipherText, string key)

* Input : Cipher text , key
* Output : Plain text

**(3)CryptAnalysis**

string Analyse(string plainText, string cipherText)

* Input : Plain text , Cipher text
* Output : key string

**Columnar :**

int key\_numOf\_Eelemnts

* Number of key elements (Cin)

text.erase(std::remove\_if(text.begin(), text.end(), ::isspace), text.end())

* Remove space from string

transform(plaintext.begin(), plaintext.end(), plaintext.begin(), ::toupper)

* Convert string to upper case

ceil(float)

* returns the smallest possible integer value which is greater than or equal to the given argument.

**(1)Encryption**

string Encrypt(string plaintext, int key[])

* Input : plain text , array of key elements
* Output : cipher text

**(2)Decryption**

string Decrypt(string ciphertext, int key[])

* Input : Cipher text , array of key elements
* Output : Plain text

**(3)CryptAnalysis**

list<int> analyse(string plaintext, string ciphertext);

* Input : Plain text , Cipher text
* Output : list of key elements

**Hill Cipher :**

int smallerdet(int row, int col, Eigen::Matrix3d M)

* Find determinant of 2x2 partial matrix of 3x3
* Input : row and column to cancel , 3x3 matrix
* Output : integer determinant

Eigen::Matrix3d modular\_inverse(Eigen::Matrix3d K)

* Find modular inverse of 3x3 matrix K
* Input : 3x3 matrix
* Output : 3x3 mod inverse matrix

Eigen::Matrix2d modular\_inverse2by2(Eigen::Matrix2d K)

* Find modular inverse of 2x2 matrix
* Input : 2x2 matrix
* Output : 2x2 mod inverse matrix

list <int> analyse3by3(list <int> plain, list <int> cipher)

* Preform cryptanalysis on 3x3 matrix plain and cipher “K=P^-1\*C”
* Input : cipher and plain as list
* Output : key matrix as list

list <int> analyse(list <int> plain, list <int> cipher)

* Preform cryptanalysis on 2x2 matrix plain and cipher “K=P^-1\*C”
* Input : cipher and plain as list
* Output : key matrix as list

list <int> encrypt(list <int> plain, list<int> key)

* Encrypt plain text using given key
* Input : list key , list plain
* Output: list cipher

list <int> decrypt(list <int> cipher, list <int> key)

* decrypt cipher text using given key
* Input : list key , list cipher
* Output: list plain

**DES :**

bool Length\_input(string txt, int size)

* To check size of key
* Input : key , default size(18 ----->0x)
* Output : boolean(True or False)

bool Check\_Hex(string txt)

* Check the hexadecimal value
* Input : string of hexadecimal
* Output : boolean (True or False)

Transform(text.begin(), text.end(), text.begin(), ::toupper)

* Convert string to upper case

string hex\_to\_bin(string sb\_wrod)

* Convert hexadecimal value to its binary
* Input : Hexa string
* Output : Binary string

void binstr\_to\_binarr(string plainText\_bin, int\* arr)

* Convert binary string to binary array
* Input : binary text , return array
* Fill array with binary text

int\*\* New\_keys(int key[])

* Create 16 sub keys from initial key
* Input : Initial key
* Output : 2D array of sub keys (16\*48)

void txtpermute(int\* pc, int\* cipherdtxt, int size1, int\* return\_arr)

* Permutatuion Function
* Input : permutation array , text , size of new array after permut , return array

void split(int\* bigarr, int bigarrSize, int\* left, int\* right, int partArrSize)

* Split text in each round to right and left side (32bit , 32bit)
* Input : text array , size of text array , left side array , right side array , 32

void Fn(int\* arr, int\* key\_arr, int\* returnedarray)

* Expansion : right (32 bit) 🡪right (48 bit)
* XOR between right side and key

void rows(int arr[], int\* row)

* Determine the row index
* Input : XORed array , return array with row indexes

void cols(int arr[], int\* cols)

* Determine the column index
* Input : XORed array , return array with column indexes

void S\_BOX(int rows[], int cols[], int\* ret\_arr)

* Store intersections of rows and columns(sbox) in array (from 48bit to 32bit)
* Input : row indexes array , columns indexes array , returned array

void dectobinary(int value, int\* binaryArr, int arrsize)

* Convert decimal value to binary array
* Input : decimal value , returned binary array , size of array

void swapp(int\* arr1, int\* arr2)

* Swap elements of two arrays
* Input : array1 , array2

void merge(int\* bigarr, int bigarrSize, int\* left, int\* right, int partArrSize)

* Merge right and left sides in one array
* Input : return merged array , 32 , left side array , right side array , 64

string bin\_to\_hex2(string key)

* Convert binary to hexa
* Input : key in binary
* Output : key in hexa

void rot(int\* X)

* Left circular shift for subkeys
* Input : array of key

**(1)Encryption**

string Encrypt(string text, string text)

* Input : hex Plain text , hex key
* Output : Cipher Text

**(2)Decryption**

string Decrypt(string text, string text)

* Input : hex Cipher text , hex key
* Output : Plain Text

**AES :**

bool Length\_input(string txt, int size)

* To check size of key
* Input : key , default size(18 ----->0x)
* Output : boolean(True or False)

bool Check\_Hex(string txt)

* Check the hexadecimal value
* Input : string of hexadecimal
* Output : boolean (True or False)

string xoring(string first, string second, int length)

* Input : first string , second string , length of xoring
* Output : XORed binary string

**(1)Key Generation**

transform(key.begin(), key.end(), key.begin(), ::tolower)

* Convert string to lower case

string rotation(string L\_word);

* Left circular shift
* Input : last word
* Output : last word after rotation

string sbox\_pick(string L\_word)

* Store intersections of rows and columns(sbox) in new string
* Input : last word
* Output :new string picked from sbox

string zerox\_rem(string sb\_word)

* Remove 0x of sbox elements from sb word
* Input : sb\_word
* Output: sb\_word without 0x (except initial 0x)

string hex\_to\_bin(string sb\_wrod)

* Convert hexadecimal value to its binary
* Input : Hexa string
* Output : Binary string

string bin\_to\_hex2(string key)

* Convert binary to hexa
* Input : key in binary
* Output : key in hexa

void key\_expansion(string key, int numOfround)

* Input : key , number of round
* Fill key\_rounds array with sub keys

**(2)Encryption**

string first\_time(string text, string text)

string Initial\_XoR(string plain, string key)

* XORing between plain text and initial key
* Input : plain text , initial key
* Output : XORed value in hexa

void shift\_rows(int Matrix\_sBox[4][4])

* Input : matrix after sub bytes
* Shifting matrix

string DEC\_To\_HEX(int dec)

* Convert decimal value to its hexa
* Input : decimal
* Output : hexa string

bool dectobinary(int decimal, int binaryArr[], int mult)

* Input : element of sub bytes , returned array , element of mix columns
* Output : boolean defines xoring with 1b array or not

void XoR\_arr(int arr[], int arr2[], int size, int\* ret\_arr)

* Binary Xoring
* Input : binary array 1 , binary array 2 , size of array , return binary array

string MIX\_arr\_calc(int temp\_arr\_mix[4], int S\_Box[4][4])

* Multiplication of mix columns matrix by sub bytes matrix
* Input : column of mix matrix , column of sub bytes matrix
* Output : hex value of Cipher Text

string AES\_Encrypt(string plaintext, string key)

* Encryption of all rounds except last one
* Input : plaint text , key string
* Output : Cipher text after xoring with key

string Lastround\_AES\_Encrypt(string plaintext, string key)

* Encryption of last round
* Input : cipher text of round n-1 , key round n
* Output : Cipher text

string AES\_Final\_ENC(string plain, string initial\_key)

* Encryption of all rounds
* Input : plaint text , initial key string
* Output : Cipher text

**(3)Decryption**

string AES\_Decrypt(string cipher, string initial\_key)

* Take initial key and find all key rounds then decrypt the cipher
* Input : initial key, cipher text
* Output: string plain text

void shift\_rows\_right(string Matrix\_sBox[4][4])

* Take the result from sbox and shif rows according to the rules
* Input : matrix sbox
* Output : no output since the array is passed by reference

void inv\_sbox(string matrix[4][4])

* Substitute the give matrix indexes with corresponding values
* Input : matrix
* Output : no output since the array is passed by reference

void inv\_mix\_col(string matrix[4][4])

* Multiply each column of matrix by the inverse of mix matrix
* Input : matrix
* Output : no output since the array is passed by reference

unsigned char inv0e(unsigned char b)

* Multiply unsigned int by 0e
* Input : unsigned char
* Output: unsigned char

unsigned char inv0d(unsigned char b)

* Multiply unsigned int by 0d
* Input : unsigned char
* Output: unsigned char

unsigned char inv09(unsigned char b)

* Multiply unsigned int by 0e
* Input : unsigned char
* Output: unsigned char

unsigned char inv0b(unsigned char b)

* Multiply unsigned int by 0e
* Input : unsigned char
* Output: unsigned char

unsigned char inv02(unsigned char b)

* Shift unsigned char by one bit ‘’multiply it by 2’’
* Input : unsigned char
* Output: unsigned char

**RSA :**

int modulo(int a, int b, int n)

* Take the base and power of number and find the modulus n
* Input : a “number” , b “base” , n “ mod n”
* Output : result of mod operation

int Encrypt(int p, int q, int M, int e)

* It encrypt integer message with rsa
* Input : p,q “prime numbers” , M “message” , e
* Output : int cipher

int gcd(int a, int b)

* Find greatest common divisor between two numbers
* Input : two numbers a&b
* Output : GCD (a,b)

int modulusinvesre(int num, int quotient)

* Find modular inverse between two numbers
* Input : two numbers
* Output : mod inverse num of quotient

int Decrypt(int p, int q, int C, int e)

* decrypt given cipher
* input : p ,q , c, e
* output : int plain

**MD5 Hash :**

string str\_to\_bin(string input)

* Input : string
* Output : binary string

string str\_padded\_to\_448(string input, int& size)

* Padding to 448 bits
* Input : string wanted to be padded , size of string (by reference 🡪calculated inside function)
* Output : padded string

string add\_length(string input, int size)

* Concate length of string with itself
* Input : padded string , length
* Output : 512 bit string

void block\_to\_32bits(string block\_512, string block\_32bits[16], int round, bool last\_block)

* Split text into blocks of 32 bits (16 blocks)
* Input : text , array of blocks , round number , last block boolean (for little endian)

string littleEndian32Bits(string str)

* Input : text
* Output : text ordered in little endian

uint32\_t G\_function(uint32\_t b, uint32\_t c, uint32\_t d, int r)

* Combination of math operations based on round number
* Input : initial vector[1] , initial vector[2], initial vector[3] , round number
* Output : unsigned 32 decimal number

void binary\_str\_to\_binary\_int\_arr(string binary, int ret\_bin\_arr[], int size)

* Convert binary string to integer array
* Input : string binary , return array , size of return array

uint32\_t binary\_dec(int arr\_bin[], int size)

* Convert binary to decimal
* Input : binary array , size of array
* Output : unsigned 32 decimal value

void CLS(int s\_round, int bits[])

* Circular left shift
* Input : round number , array of integer
* Store in bits arrat after shifting

void shifting(int bits[], int num\_shifts);

* Circular left shift , Stored in same array
* Input : array of integer , number of shifts based on round number
* Calling inside CLS function

string decToHexa(uint32\_t n)

* Input : unsigned 32 decimal value
* Output : hexa string

string littleEndian\_hex(string str)

* Input : hexa string
* Output : hexa string ordered in little endian

uint32\_t HexToDec(string n)

* Input : Hex string
* Output : unsigned 32 decimal value

string repeated\_16(string block\_512, int round\_num, bool last\_block)

* Input : block 512 bits , round number , boolean last block(for little endian)
* Output : CVi for next round (4 rounds)

string MD5\_Hash(string plaintext)

* Input : plaintext
* Output : hashing