22BCE3799

Apurba Koirala

Cryptography and Network Security Lab Assessment 6

1. Caesar Cipher

#include <iostream>

#include <cmath>

using namespace std;

string encrypt(string text, int s) {

string result = "";

for (int i = 0; i < text.length(); i++) {

if (isupper(text[i])) {

result += char(int(text[i] + s - 65) % 26 + 65);

} else {

result += char(int(text[i] + s - 97) % 26 + 97);

}

}

return result;

}

string decrypt(string text, int s) {

string result = "";

for (int i = 0; i < text.length(); i++) {

if (isupper(text[i])) {

result += char(int(text[i] - s - 65 + 26) % 26 + 65);

} else {

result += char(int(text[i] - s - 97 + 26) % 26 + 97);

}

}

return result;

}

int main() {

string text;

int s;

cin >> text;

cin >> s;

cout << "Text: " << text << "\n";

cout << "Shift: " << s << "\n";

string encrypted\_text = encrypt(text, s);

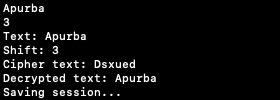
cout << "Cipher text: " << encrypted\_text << "\n";

cout << "Decrypted text: " << decrypt(encrypted\_text, s);

return 0;

}

Output:



1. Hill Cipher(Encryption Only)

Code:

#include <iostream>

#include <vector>

#include <string>

using namespace std;

vector<int> performMatrixMultiplication(vector<vector<int> > &matrix,

vector<int> &vectorInput, int mod) {

int size = matrix.size();

vector<int> result(size, 0);

for (int row = 0; row < size; row++) {

for (int col = 0; col < size; col++) {

result[row] += matrix[row][col] \* vectorInput[col];

}

result[row] = (result[row] % mod + mod) % mod;

}

return result;

}

string encryptMessage(string plainText, vector<vector<int> > &keyMatrix, int matrixSize) {

string encryptedText = "";

while (plainText.size() % matrixSize != 0) {

plainText += 'X';

}

for (size\_t i = 0; i < plainText.size(); i += matrixSize) {

vector<int> letterVector(matrixSize);

for (int j = 0; j < matrixSize; j++) {

letterVector[j] = plainText[i + j] - 'A';

}

vector<int> encryptedVector = performMatrixMultiplication(keyMatrix, letterVector, 26);

for (int j = 0; j < matrixSize; j++) {

encryptedText += (encryptedVector[j] + 'A');

}

}

return encryptedText;

}

int main() {

string plainText;

cout << "Enter the message to encrypt (uppercase letters only): ";

cin >> plainText;

int matrixSize;

cout << "Enter key matrix size: ";

cin >> matrixSize;

vector<vector<int> > encryptionKey(matrixSize, vector<int>(matrixSize));

cout << "Enter the elements of key matrix:\n";

for (int i = 0; i < matrixSize; i++) {

for (int j = 0; j < matrixSize; j++) {

cin >> encryptionKey[i][j];

}

}

string encryptedText = encryptMessage(plainText, encryptionKey, matrixSize);

cout << "Encrypted message: " << encryptedText << endl;

return 0;

}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

1. Vernam Cipher

Code:

#include <iostream>

#include <string>

using namespace std;

string generateKey(string text, string key) {

int textLength = text.size();

int keyLength = key.size();

for (int i = 0; key.size() < textLength; i++) {

key.push\_back(key[i % keyLength]);

}

return key;

}

string toUpperCase(string str) {

for (size\_t i = 0; i < str.size(); i++) {

if (str[i] >= 'a' && str[i] <= 'z') {

str[i] = str[i] - 'a' + 'A';

}

}

return str;

}

string encryptText(string text, string key) {

string cipherText;

for (size\_t i = 0; i < text.size(); i++) {

char encryptedChar = (text[i] + key[i] - 2 \* 'A') % 26 + 'A';

cipherText.push\_back(encryptedChar);

}

return cipherText;

}

string decryptText(string cipherText, string key) {

string originalText;

for (size\_t i = 0; i < cipherText.size(); i++) {

char decryptedChar = (cipherText[i] - key[i] + 26) % 26 + 'A';

originalText.push\_back(decryptedChar);

}

return originalText;

}

int main() {

string text, keyword;

cout << "Enter the text: ";

cin >> text;

cout << "Enter the keyword: ";

cin >> keyword;

text = toUpperCase(text);

keyword = toUpperCase(keyword);

string key = generateKey(text, keyword);

cout << "Generated Key: " << key << endl;

string cipherText = encryptText(text, key);

cout << "Encrypted Text: " << cipherText << endl;

string originalText = decryptText(cipherText, key);

cout << "Decrypted Text: " << originalText << endl;

return 0;

}

Output:

A black screen with white text

AI-generated content may be incorrect.

1. Rail Fence Cipher(Encryption only)

Code:

#include <iostream>

#include <string>

using namespace std;

string encryptRailFence(string text, int key) {

char rail[key][text.length()];

for (int i = 0; i < key; i++)

for (int j = 0; j < text.length(); j++)

rail[i][j] = '\n';

bool dir\_down = false;

int row = 0, col = 0;

for (int i = 0; i < text.length(); i++) {

if (row == 0 || row == key - 1)

dir\_down = !dir\_down;

rail[row][col++] = text[i];

dir\_down ? row++ : row--;

}

string result;

for (int i = 0; i < key; i++)

for (int j = 0; j < text.length(); j++)

if (rail[i][j] != '\n')

result.push\_back(rail[i][j]);

return result;

}

int main() {

string text;

int key;

cout << "Enter the text: ";

getline(cin, text);

cout << "Enter key: ";

cin >> key;

cout << "Encrypted text: " << encryptRailFence(text, key) << endl;

return 0;

}

Output:

A black background with white text

AI-generated content may be incorrect.

1. Fermats theorem:

#include <iostream>

#include <cmath>

using namespace std;

bool prime(int p){

if(p<=1){

return false;

}

for (int i = 2; i <= sqrt(p); i++){

if(p%i == 0){

return false;

}

}

return true;

}

int gcd(int a, int b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

void fermats(int a, int b, int p){

if (b == p-1){

cout<<"Answer = "<<1;

return;

}

else if(b == p){

cout<<"Answer = "<<a;

return;

}

else{

cout<<a<<"^"<<b<<" MOD "<<p<<"\n";

int quotient = b / p;

int rem = b % p;

if (quotient+rem < p){

cout<<a<<"^"<<(quotient+rem)<<" MOD "<<p<<"\n";

double result = pow(a, b);

cout<<"Answer = "<< fmod(result, p)<<"\n";

}

else{

fermats(a, quotient+rem, p);

}

}

}

int main()

{

int a, b, p;

cout<< "enter a^b mod p values: ";

cin>>a;

cin>>b;

cin>>p;

if (!prime(p)){

cout<<"fermats theorem not applicable";

return 1;

}

else if(gcd(a, p) != 1){

cout<<"fermats theorem not applicable";

return 1;

}

else{

cout<<"conditions satisfied for fermats theorem.\n";

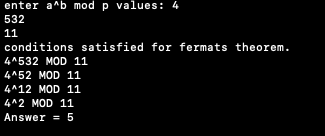
fermats(a, b, p);

}

return 0;

}

Output:



1. Eulers Theorem

Code:

#include <iostream>

#include <cmath>

using namespace std;

bool isPrime(int num) {

if (num < 2) return false;

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) return false;

}

return true;

}

int gcd(int a, int b) {

if (b == 0) return a;

return gcd(b, a % b);

}

int getPhi(int n) {

if (isPrime(n)) {

return n - 1;

}

for (int p = 2; p \* p <= n; p++) {

if (n % p == 0) {

int q = n / p;

if (isPrime(q) && isPrime(p) && p != q) {

cout << "p = " << p << "\tq = " << q << "\n";

return (p - 1) \* (q - 1);

}

}

}

int result = n;

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0) {

while (n % i == 0) {

n /= i;

}

result -= result / i;

}

}

if (n > 1) {

result -= result / n;

}

return result;

}

void eulers(int a, int b, int n) {

int phin = getPhi(n);

cout << "phi(n) = " << phin << "\n";

int rem = b % phin;

int result = pow(a, rem);

int answer = fmod(result, n);

cout << "Answer = " << answer << endl;

}

int main() {

int a, b, n;

cout << "Enter a, b, and n to compute a^b MOD n: ";

cin >> a >> b >> n;

if (gcd(a, n) != 1) {

cout << "Euler's theorem cannot be applied because gcd(a, n) != 1.\n";

return 1;

}

eulers(a, b, n);

return 0;

}

Output:

For all three cases.

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

1. Diffie Hellman Key Exchange with middle attack

Code:

#include <iostream>

#include <cmath>

using namespace std;

int main(){

int g, n, xa, A, yb, B, xc, yc, Ac, Bc, K1, K2, K1p, K2p;

cout<<"Enter g and n:\n";

cin>>g>>n;

cout<<"Enter private key of A\n";

cin>>xa;

A = (static\_cast<long long>(pow(g, xa))) % n;

cout<<"Enter private key of B\n";

cin>>yb;

B = (static\_cast<long long>(pow(g, yb))) % n;

cout<<"A: "<<A<<" B: "<<B<<"\n";

cout<<"Enter x and y used by the attacker:\n";

cin>>xc>>yc;

cout<<"A and B values calculated by the attacker: ";

Ac = (static\_cast<long long>(pow(g, xc))) % n;

Bc = (static\_cast<long long>(pow(g, yc))) % n;

cout<<"Ac: "<<Ac<<" Bc: "<<Bc<<"\n";

cout<<"K1 and K2 values calculated by the attacker: ";

K1 = (static\_cast<long long>(pow(Bc, xa))) % n;

K2 = (static\_cast<long long>(pow(Ac, yb))) % n;

cout<<"K1: "<<K1<<" K2: "<<K2<<"\n";

cout<<"K1\* and K2\* values calculated by the attacker: ";

K1p = (static\_cast<long long>(pow(B, xc))) % n;

K2p = (static\_cast<long long>(pow(A, yc))) % n;

cout<<"K1\*: "<<K1p<<" K2\*: "<<K2p<<"\n";

if(K1p == K2 && K2p == K1){

cout<<"Man in the middle attack shown in diffie hellman as K1 = K2\* and K2 = K1\*";

}

else{

cout<<"Attack failed";

}

return 0;

}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

1. GCD using Extended Euclidean Algorithm

Code:

#include <iostream>

#include <cmath>

using namespace std;

int extended\_euc(int r1, int r2, int s1, int s2, int t1, int t2){

int q = r1/r2;

int r = r1 % r2;

int s = s1 - s2 \* q;

int t = t1 - t2 \* q;

if (r == 0){

return r2;

}

return extended\_euc(r2, r, s2, s, r2, t);

}

int main(){

int r1, r2, s1, s2, t1, t2;

t2, s1 = 1;

s2, t1 = 0;

cout<<"Enter a and b for gcd(a, b): \n";

cin>> r1>> r2;

cout<<"gcd("<<r1<<", "<<r2<<")\n";

int answer = extended\_euc(r1, r2, s1, s2, t1, t2);

cout<<answer;

}

Output:

Same as class example:

A black screen with white text

AI-generated content may be incorrect.

1. Initial Permutation in DES

Code:

#include <iostream>

#include <bitset>

using namespace std;

int IP[] = {

58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7

};

bitset<64> initialPermutation(bitset<64> input) {

bitset<64> permuted;

for (int i = 0; i < 64; i++) {

permuted[63 - i] = input[64 - IP[i]];

}

return permuted;

}

int main() {

uint64\_t input;

cout << "Enter a 64-bit number (in hex): ";

cin >> hex >> input;

bitset<64> inputBits(input);

cout << "Original 64-bit input: " << inputBits << endl;

bitset<64> permutedBits = initialPermutation(inputBits);

cout << "After Initial Permutation: " << permutedBits << endl;

return 0;

}

Output:



1. RC4 Stream Generation

Code:

#include <iostream>

#include <vector>

using namespace std;

void printS(const vector<int>& S) {

for (int val : S) {

cout << val << " ";

}

cout << endl;

}

void rc4\_keystream(vector<int>& S, int numBytes) {

int i = 0, j = 0;

cout << "Generated Keystream:\n";

for (int count = 0; count < numBytes; count++) {

i = (i + 1) % 8;

j = (j + S[i]) % 8;

swap(S[i], S[j]);

int t = (S[i] + S[j]) % 8;

int k = S[t];

cout << k << " ";

}

cout << endl;

}

int main() {

vector<int> S(8);

cout<<"Enter S vector: \n";

for (int i = 0; i<8; i++){

cin>>S[i];

}

cout << "Initial S: ";

printS(S);

int numBytes;

cout << "Enter number of keys to generate: ";

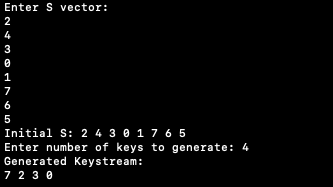
cin >> numBytes;

rc4\_keystream(S, numBytes);

return 0;

}

Output:



1. RSA Encryption and Decryption

Code:

#include <iostream>

#include <cmath>

using namespace std;

bool isPrime(int num) {

if (num < 2) return false;

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) return false;

}

return true;

}

int getPhi(int N, int P, int Q){

int result = N;

if(isPrime(P) && isPrime(Q)){

return (P-1) \* (Q-1);

}

else{

for (int i = 2; i \* i <= N; i++){

while (N % i == 0){

N /= i;

}

result = result - (result/i);

}

}

if(N > 1){

result -= result / N;

}

return result;

}

int Eulers(int CT, int D, int N, int P, int Q) {

int phin = getPhi(N, P, Q);

int rem = D % phin;

//used successive squaring

int result = 1;

for (int i = 0; i < rem; i++) {

result = (result \* CT) % N;

}

return result;

}

int checkNeg(int t2, int phi){

if (t2 > 0){

return t2;

}

else{

int temp = abs(t2);

int m = temp % phi;

t2 = (phi - m) % phi;

return t2;

}

}

int calculateD(int phi, int r1, int r2, int t1, int t2){

int r = r1 % r2;

int q = r1/r2;

int t = t1 - t2 \* q;

if (r2 == 1 && r == 0){

t2 = checkNeg(t2, phi);

return t2;

}

else{

return calculateD(phi, r2, r, t2, t);

}

}

int main(){

int P, Q;

cout<<"Enter two prime number, P and Q: \n";

cin>>P;

cin>>Q;

int N = P \* Q;

cout<<"N: "<<N<<"\n";

int phi = getPhi(N, P, Q);

cout<<"Enter Public Key E for Encryption: \n";

int E;

cin>>E;

cout<<"Finding private key D for decryption using formula (D \* E) MOD phi(N) = 1\n";

int D;

D = calculateD(phi, phi, E, 0, 1);

cout<<"Calculated D value: "<<D<<"\n";

long long CT, PT;

cout<<"Enter Message to be encrypted such that it is smaller than P \* Q: \n";

cin>>PT;

CT =

cout<<"Cipher Text: "<< CT<<"\nDecrypting:\n";

PT = Eulers(CT, D, N, P, Q);

cout<<"Decrypted Text: "<< PT;

}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

1. Point Doubling in ECC

Code:

#include <iostream>

#include <cmath>

using namespace std;

int negcheck(int p, int t2){

int temp = abs(t2) % p;

return (p - temp);

}

int inverse(int p, int r1, int r2, int t1, int t2){

int r = r1 % r2;

int q = r1/r2;

int t = t1 - t2 \* q;

if(r2 == 1 && r == 0){

if(t2<0){

t2 = negcheck(p, t2);

return t2;

}

else{

return t2;

}

} else{

return inverse(p, r2, r, t2, t);

}

}

int main(){

int p, a, x, y;

cout<<"Enter p, a, x and y values sequentially:\n";

cin>>p>>a>>x>>y;

int up = ((3 \* x \* x) + a);

int down = (2\*y);

down = inverse(p, p, down, 0, 1);

int lambda = (up \* down) % p;

int x3 = (lambda\*lambda - x - x) % p;

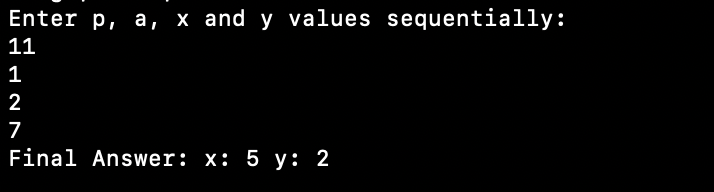
int y3 = (negcheck(p, lambda\*(x - x3) - y)) % p;

cout<<"Final Answer: x: "<<x3<<" y: "<< y3<<"\n";

return 0;

}

Output:



1. Negative Point in ECC

Code:

#include <iostream>

#include <cmath>

using namespace std;

int negcheck(int y, int q){

int temp = y % q;

return (q - temp);

}

int main(){

int x, y;

cout<<"Enter x and y\n";

cin>>x;

cin>>y;

cout<<"Negative point: x, -y % q\n";

cout<<"Enter q a prime number: \n";

int q;

cin>>q;

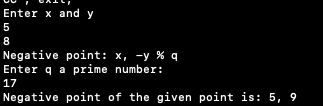
y = negcheck(y, q);

cout<<"Negative point of the given point is: "<<x<<", "<<y;

return 0;

}

Output:



1. Elgamal Cryptography

Code:

#include <iostream>

#include <cmath>

using namespace std;

int calculateYa(int a, int Xa, int q){

return (static\_cast<long long>(pow(a, Xa)))%q;

}

bool primitive(int q, int a) {

if (q <= 1 || a <= 1 || a >= q) {

return false;

}

for (int i = 1; i < q; i++) {

if (pow(a, i) - (int)(pow(a, i) / q) \* q == 0) {

return false;

}

}

return true;

}

int calculateK(int Ya, int k, int q){

return (static\_cast<long long>(pow(Ya, k)))%q;

}

int calculateC1(int a, int k, int q){

return (static\_cast<long long>(pow(a, k)))%q;

}

int calculateC2(int K, int M, int q){

return (K \* M) % q;

}

int checkNeg(int t2, int phi){

if (t2 > 0){

return t2;

}

else{

int temp = abs(t2);

int m = temp % phi;

t2 = (phi - m) % phi;

return t2;

}

}

int inverse(int q, int r1, int r2, int t1, int t2){

int r = r1 % r2;

int qa = r1/r2;

int t = t1 - t2 \* qa;

if (r2 == 1 && r == 0){

t2 = checkNeg(t2, q);

return t2;

}

else{

return inverse(q, r2, r, t2, t);

}

}

int main(){

int q, a;

cout<<"Enter prime numbers q and a, such that a is a primitive root of q: \n";

cin>>q;

cin>>a;

if(!primitive(q, a)){

return 1;

}

int Xa, Ya;

cout<<"Enter private key: \n";

cin>>Xa;

Ya = calculateYa(a, Xa, q);

cout<<"Ya: "<<Ya<<"\n";

cout<<"Choose k\n";

int k;

cin>>k;

int K = calculateK(Ya, k, q);

int c1, c2, M;

cout<<"Enter Message: \n";

cin>>M;

c1 = calculateC1(a, k, q);

c2 = calculateC2(K, M, q);

cout<<"C1 and C2: ("<<c1<<", "<<c2<<")\n"<<"Decryption:\n"<<"Recover K\n";

K = calculateK(c1, Xa, q);

cout<<"K: "<<K<<"\n"<<"Recovering M:\n"<< "M:" ;

M = (c2 \* inverse(q, q, K, 0, 1))% q;

cout<<M<<"\n";

return 0;

}

Output:

A black screen with white text

AI-generated content may be incorrect.