1. **Design and implement a product cipher using substitution and transposition ciphers.**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

string caesar(const string &text, int shift) {

    string res = "";

    for (char ch : text)

        res += isalpha(ch) ? (char)((ch - (isupper(ch) ? 'A' : 'a') + shift + 26) % 26 + (isupper(ch) ? 'A' : 'a')) : ch;

    return res;

}

string railEncrypt(const string &text, int rails) {

    vector<string> rail(rails);

    int row = 0, dir = 1;

    for (char ch : text) {

        rail[row] += ch;

        if (row == 0) dir = 1;

        else if (row == rails - 1) dir = -1;

        row += dir;

    }

    string res = "";

    for (auto &r : rail) res += r;

    return res;

}

string railDecrypt(const string &cipher, int rails) {

    vector<int> len(rails, 0);

    int row = 0, dir = 1, n = cipher.size();

    for (int i = 0; i < n; ++i) {

        len[row]++;

        if (row == 0) dir = 1;

        else if (row == rails - 1) dir = -1;

        row += dir;

    }

    vector<string> rail(rails);

    int idx = 0;

    for (int r = 0; r < rails; ++r) {

        rail[r] = cipher.substr(idx, len[r]);

        idx += len[r];

    }

    string res = "";

    row = 0, dir = 1;

    vector<int> pos(rails, 0);

    for (int i = 0; i < n; ++i) {

        res += rail[row][pos[row]++];

        if (row == 0) dir = 1;

        else if (row == rails - 1) dir = -1;

        row += dir;

    }

    return res;

}

int main() {

    string mode, msg, res;

    int shift, rails;

    cout << "Mode (encrypt/decrypt): "; cin >> mode; cin.ignore();

    cout << "Message: "; getline(cin, msg);

    cout << "Shift: "; cin >> shift;

    cout << "Rails: "; cin >> rails;

    msg.erase(remove(msg.begin(), msg.end(), ' '), msg.end());

    if (mode == "encrypt")

        res = railEncrypt(caesar(msg, shift), rails);

    else if (mode == "decrypt")

        res = caesar(railDecrypt(msg, rails), -shift);

    else

        res = "Invalid mode!";

    cout << "\nResult: " << res << endl;

}

**OUTPUT**

PS S:\WorkSpace\Academics\SEM-6\Cryptographics> ./a

Mode (encrypt/decrypt): encrypt

Message: Hello World

Shift: 23

Rails: 7

Result: EbiialiTol

PS S:\WorkSpace\Academics\SEM-6\Cryptographics> ./a

Mode (encrypt/decrypt): decrypt

Message: EbiialiTol

Shift: 23

Rails: 7

Result: HelloWorld

1. **Implement encryption and decryption of the affine cipher.**

#include <iostream>

#include <string>

#include <cctype>

using namespace std;

int modInv(int a, int m) {

    for (int x = 1; x < m; ++x) if ((a \* x) % m == 1) return x;

    return -1;

}

string affine(string text, int a, int b, char mode) {

    string res = "";

    int a\_inv = (mode == 'd') ? modInv(a, 26) : 0;

    if (mode == 'd' && a\_inv == -1) return "Invalid 'a' (no inverse).";

    for (char c : text) {

        if (isalpha(c)) {

            char base = isupper(c) ? 'A' : 'a';

            int x = c - base;

            int v = (mode == 'e') ? (a \* x + b) % 26 : (a\_inv \* (x - b + 26)) % 26;

            res += char(v + base);

        } else res += c;

    }

    return res;

}

int main() {

    string text; int a, b; char mode;

    cout << "Message: "; getline(cin, text);

    cout << "Key a (coprime with 26): "; cin >> a;

    cout << "Key b: "; cin >> b;

    cout << "Mode (e/d): "; cin >> mode;

    cout << "\nResult: " << affine(text, a, b, tolower(mode)) << endl;

}

**OUTPUT**

PS S:\WorkSpace\Academics\SEM-6\Cryptographics> ./a

Message: Do you like this world

Key a (coprime with 26): 9

Key b: 6

Mode (e/d): e

Result: Hc oce basq vram wcdbh

PS S:\WorkSpace\Academics\SEM-6\Cryptographics> ./a

Message: Hc oce basq vram wcdbh

Key a (coprime with 26): 9

Key b: 6

Mode (e/d): d

Result: Do you like this world

1. **Implement Diffie-Hellman Key Exchange Algorithm.**

#include <iostream>

using namespace std;

long long mod\_exp(long long base, long long exp, long long mod) {

    long long res = 1;

    base %= mod;

    while (exp) {

        if (exp & 1) res = res \* base % mod;

        base = base \* base % mod;

        exp >>= 1;

    }

    return res;

}

int main() {

    long long p, g, a, b;

    cout << "Prime p: "; cin >> p;

    cout << "Primitive root g: "; cin >> g;

    cout << "Alice's private key: "; cin >> a;

    cout << "Bob's private key: "; cin >> b;

    long long A = mod\_exp(g, a, p);

    long long B = mod\_exp(g, b, p);

    long long s1 = mod\_exp(B, a, p);

    long long s2 = mod\_exp(A, b, p);

    cout << "\nAlice's Public Key: " << A

         << "\nBob's Public Key: " << B

         << "\nAlice's Shared Secret: " << s1

         << "\nBob's Shared Secret: " << s2

         << "\n" << (s1 == s2 ? "Key Match!" : "Mismatch!") << endl;

    return 0;

}

**OUTPUT**

PS S:\WorkSpace\Academics\SEM-6\Cryptographics> ./a

Prime p: 103

Primitive root g: 3

Alice's private key: 6

Bob's private key: 2

Alice's Public Key: 8

Bob's Public Key: 9

Alice's Shared Secret: 64

Bob's Shared Secret: 64

Key Match!

1. **Implement RSA Public Key Cryptosystem.**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <cmath>

using namespace std;

int gcd(int a, int b) {

    return b == 0 ? a : gcd(b, a % b);

}

bool is\_prime(int n) {

    if (n < 2) return false;

    if (n < 4) return true;

    if (n % 2 == 0 || n % 3 == 0) return false;

    for (int i = 5; i \* i <= n; i += 6)

        if (n % i == 0 || n % (i + 2) == 0)

            return false;

    return true;

}

int generate\_prime(int min\_val = 100, int max\_val = 300) {

    while (true) {

        int p = min\_val + rand() % (max\_val - min\_val);

        if (is\_prime(p)) return p;

    }

}

int mod\_inverse(int e, int phi) {

    int a = e, b = phi, x0 = 1, x1 = 0;

    while (b) {

        int q = a / b, temp = b;

        b = a % b, a = temp;

        temp = x1;

        x1 = x0 - q \* x1, x0 = temp;

    }

    return (x0 + phi) % phi;

}

int mod\_exp(int base, int exp, int mod) {

    int result = 1;

    while (exp) {

        if (exp % 2) result = (result \* base) % mod;

        base = (base \* base) % mod;

        exp /= 2;

    }

    return result;

}

void generate\_keys(pair<int, int>& pub, pair<int, int>& priv) {

    int p = generate\_prime(), q;

    do { q = generate\_prime(); } while (q == p);

    int n = p \* q, phi = (p - 1) \* (q - 1), e = 2 + rand() % (phi - 2);

    while (gcd(e, phi) != 1) e = 2 + rand() % (phi - 2);

    pub = {e, n}, priv = {mod\_inverse(e, phi), n};

}

vector<int> encrypt(const string& text, pair<int, int> pub) {

    vector<int> cipher;

    for (char ch : text) cipher.push\_back(mod\_exp(ch, pub.first, pub.second));

    return cipher;

}

string decrypt(const vector<int>& cipher, pair<int, int> priv) {

    string text;

    for (int val : cipher) text += char(mod\_exp(val, priv.first, priv.second));

    return text;

}

int main() {

    srand(time(0));

    pair<int, int> pub, priv;

    generate\_keys(pub, priv);

    cout << "RSA Key Generation\n";

    cout << "Public Key (e, n): (" << pub.first << ", " << pub.second << ")\n";

    cout << "Private Key (d, n): (" << priv.first << ", " << priv.second << ")\n";

    cout << "\nEnter a message to encrypt: ";

    string message;

    getline(cin, message);

    vector<int> encrypted = encrypt(message, pub);

    cout << "\nEncrypted: ";

    for (int val : encrypted) cout << val << " ";

    cout << "\nDecrypted: " << decrypt(encrypted, priv) << "\n";

    return 0;

}

**OUTPUT**

RSA Key Generation

Public Key (e, n): (14737, 31831)

Private Key (d, n): (28033, 31831)

Enter a message to encrypt: I'm Death

Encrypted: 3012 8678 5161 10526 27235 13370 10383 28586 22255

Decrypted: I'm Death

1. **WAP to encrypt a message using a given P-box.**

#include <iostream>

#include <vector>

#include <sstream>

using namespace std;

string pbox\_encrypt(const string &message, const vector<int> &pbox) {

    int size = pbox.size();

    string padded = message + string((size - message.length() % size) % size, ' ');

    string encrypted = padded;

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

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

            encrypted[i + j] = padded[i + pbox[j]];

        }

    }

    return encrypted;

}

string pbox\_decrypt(const string &ciphertext, const vector<int> &pbox) {

    int size = pbox.size();

    vector<int> inverse(size);

    for (int i = 0; i < size; ++i){

        inverse[pbox[i]] = i;

    }

    string decrypted = ciphertext;

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

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

            decrypted[i + j] = ciphertext[i + inverse[j]];

        }

    }

    return decrypted;

}

int main() {

    string message;

    vector<int> pbox;

    cout << "Enter the message: ";

    getline(cin, message);

    cout << "Enter P-box (0-based, space-separated): ";

    string pbox\_input;

    getline(cin, pbox\_input);

    istringstream iss(pbox\_input);

    int index;

    while (iss >> index){

        pbox.push\_back(index);

    }

    if (pbox.empty()) {

        cout << "Error: P-box cannot be empty!\n";

        return 1;

    }

    string encrypted = pbox\_encrypt(message, pbox);

    string decrypted = pbox\_decrypt(encrypted, pbox);

    cout << "\nEncrypted: '" << encrypted << "'\n";

    cout << "Decrypted: '" << decrypted << "'\n";

    return 0;

}

**OUTPUT**

PS S:\WorkSpace\Academics\SEM-6\Cryptographics> g++ '.\7. pbox.cpp'

PS S:\WorkSpace\Academics\SEM-6\Cryptographics> ./a

Enter the message: Internal Pointer Variable

Enter P-box (0-based, space-separated): 4 2 3 0 1

Encrypted: 'rteInPl naentoirVar eblia'

Decrypted: 'Internal Pointer Variable'