192110492
A.REVANTH
PROGRAM -1

1	ENCRYPTION ALGORITHM	Create a program to generate digital signatures for a given message using RSA. • Verify the authenticity of a message using the generated digital signature. • Test the program with various messages and keys. How • reliable is the digital signature scheme?	12.00 – 12.30 PM

AIM: Create a program to generate digital signatures for a given

message using RSA.

• Verify the authenticity of a message using the generated

digital signature.

• Test the program with various messages and keys. How

• reliable is the digital signature scheme?

ALGORITHM:

1. Take the message and the key (an integer) as input.
2. Declare a character array to store the message and read the message from the user and store it in the array.
3. Iterate through each character in the message.
4. For each alphabetic character (a-z or A-Z), shift it by the key value.
5. Print the encrypted message.
6. Take the encrypted message and the key as input.
7. Declare a character array to store the decrypted message and read the encrypted message from the user and store it in the array.
8. Iterate through each character in the encrypted message.
9. For each alphabetic character (a-z or A-Z), shift it back by the key value.
10. Print the decrypted message.

PROGRAM:

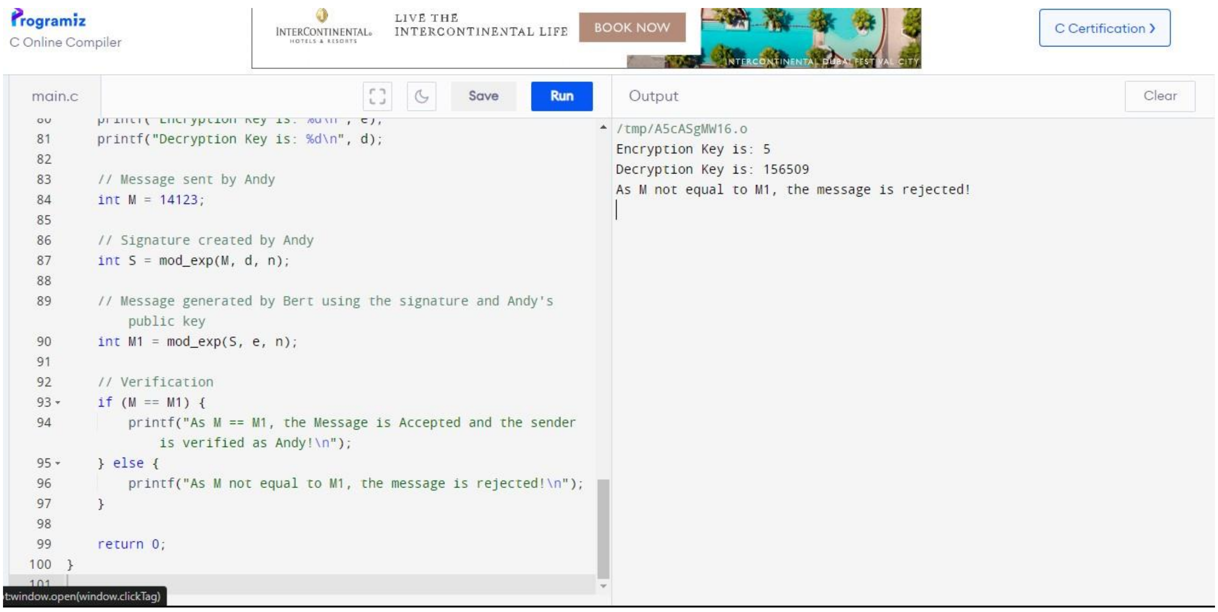
```
#include <stdio.h>

// Function to compute the greatest common divisor (gcd) of two numbers
```

```
int gcd(int p, int q) {
   if (q == 0)
      return p;
   else
      return gcd(q, p % q);
}

// Function to compute the multiplicative inverse using the extended Euclidean algorithm
int exteuclid(int a, int b) {
   int r_1 = a, r_2 = b, s_1 = 1, s_2 = 0, t_1 = 0, t_2 = 1;
   int temp, r, s, t;

   while (r_2 > 0) {
      temp = r_1 / r_2;
      r = r_1 - temp * r_2;
      r_1 = r_2;
      r_2 = r;

      s = s_1 - temp * s_2;
      s_1 = s_2;
      s_2 = s;

      t = t_1 - temp * t_2;
      t_1 = t_2;
      t_2 = t;
   }
```

```
    if (t_1 < 0)
        t_1 = t_1 + a;

    return t_1;
}

// Function to perform modular exponentiation
int mod_exp(int base, int exp, int mod) {
    int result = 1;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp = exp / 2;
    }
    return result;
}

int main() {
    int p = 823, q = 953;
    int n = p * q;
    int Pn = (p - 1) * (q - 1);

    int possible_key[Pn];
    int count = 0;

    // Generate possible encryption keys
```

```
for (int i = 2; i < Pn; i++) {
    if (gcd(Pn, i) == 1) {
        possible_key[count++] = i;
    }
}

// Select encryption key and compute its multiplicative inverse
int e = -1;
int d;
for (int i = 0; i < count; i++) {
    d = exteuclid(Pn, possible_key[i]);
    if (d > 0) {
        e = possible_key[i];
        break;
    }
}

if (e == -1) {
    printf("No possible encryption key!\n");
    return 1;
}

printf("Encryption Key is: %d\n", e);
printf("Decryption Key is: %d\n", d);

// Message sent by Andy
int M = 14123;
```

```
    // Signature created by Andy
    int S = mod_exp(M, d, n);

    // Message generated by Bert using the signature and Andy's public key
    int M1 = mod_exp(S, e, n);

    // Verification
    if (M == M1) {
        printf("As M == M1, the Message is Accepted and the sender is verified as Andy!\n");
    } else {
        printf("As M not equal to M1, the message is rejected!\n");
    }

    return 0;
}
```



2. PROGRAM -2

2	KEY EXCHANGE	Implement the Diffie-Hellman key exchange algorithm in a program. • Test the program with different prime numbers and primitive roots. How secure is the key exchange process? • Develop a program that encrypts and decrypts messages using a block cipher like AES. • Experiment with different key sizes and block sizes. How does changing these parameters affect the security and performance of the cipher?	12.30 – 1.00 PM

ALGORITHM:

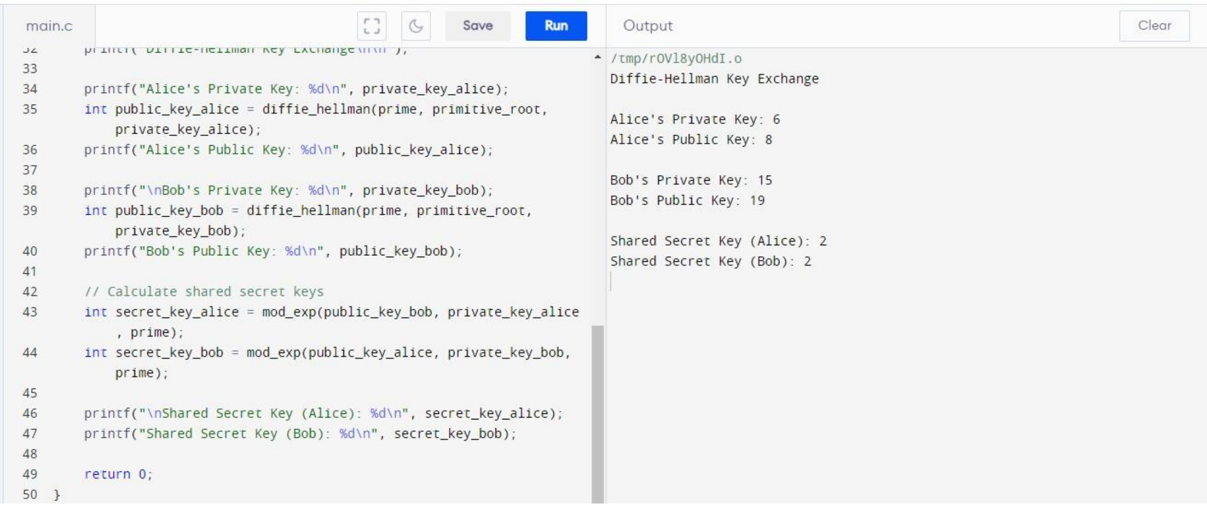
1. Take the message and the key (an integer) as input.
2. Declare a character array to store the message and read the message from the user and store it in the array.
3. Iterate through each character in the message.
4. For each alphabetic character (a-z or A-Z), shift it by the key value.
5. Print the encrypted message.
6. Take the encrypted message and the key as input.
7. Declare a character array to store the decrypted message and read the encrypted message from the user and store it in the array.
8. Iterate through each character in the encrypted message.
9. For each alphabetic character (a-z or A-Z), shift it back by the key value.
10. Print the decrypted message.

PROGRAM

```
#include <stdio.h>

#include <math.h>



// Function to calculate modular exponentiation (base^exp mod mod)

int mod_exp(int base, int exp, int mod) {

   int result = 1;

   base = base % mod;

   while (exp > 0) {

      if (exp % 2 == 1)

         result = (result * base) % mod;

      exp = exp / 2;

      base = (base * base) % mod;

   }
```

```c
    return result;
}

// Function to perform Diffie-Hellman key exchange and return public key
int diffie_hellman(int prime, int primitive_root, int private_key) {
    // Calculate public key: public_key = primitive_root^private_key mod prime
    return mod_exp(primitive_root, private_key, prime);
}

int main() {
    // Shared prime number and primitive root
    int prime = 23;
    int primitive_root = 5;

    // Private keys for Alice and Bob
    int private_key_alice = 6; // Chosen randomly
    int private_key_bob = 15; // Chosen randomly

    printf("Diffie-Hellman Key Exchange\n\n");

    printf("Alice's Private Key: %d\n", private_key_alice);
    int public_key_alice = diffie_hellman(prime, primitive_root, private_key_alice);
    printf("Alice's Public Key: %d\n", public_key_alice);

    printf("\nBob's Private Key: %d\n", private_key_bob);
    int public_key_bob = diffie_hellman(prime, primitive_root, private_key_bob);
    printf("Bob's Public Key: %d\n", public_key_bob);
```

```
    // Calculate shared secret keys

    int secret_key_alice = mod_exp(public_key_bob, private_key_alice, prime);

    int secret_key_bob = mod_exp(public_key_alice, private_key_bob, prime);


    printf("\nShared Secret Key (Alice): %d\n", secret_key_alice);

    printf("Shared Secret Key (Bob): %d\n", secret_key_bob);


    return 0;

}
```



3. PROGRAM -3

		and performance of the cipher?	
3	KEY MANAGEMET TECHNIQUE	Create a program that manages cryptographic keys securely, including generation, storage, distribution, and revocation. Explore different key management techniques such as key rotation and key escrow. What are the best practices for key management in cryptographic systems?	1.00 – 1.30 PM

ALGORITHM:

1. Take the message and the key (an integer) as input.
2. Declare a character array to store the message and read the message from the user and store it in the array.
3. Iterate through each character in the message.

4. For each alphabetic character (a-z or A-Z), shift it by the key value.
5. Print the encrypted message.
6. Take the encrypted message and the key as input.
7. Declare a character array to store the decrypted message and read the encrypted message from the user and store it in the array.
8. Iterate through each character in the encrypted message.
9. For each alphabetic character (a-z or A-Z), shift it back by the key value.
10. Print the decrypted message.

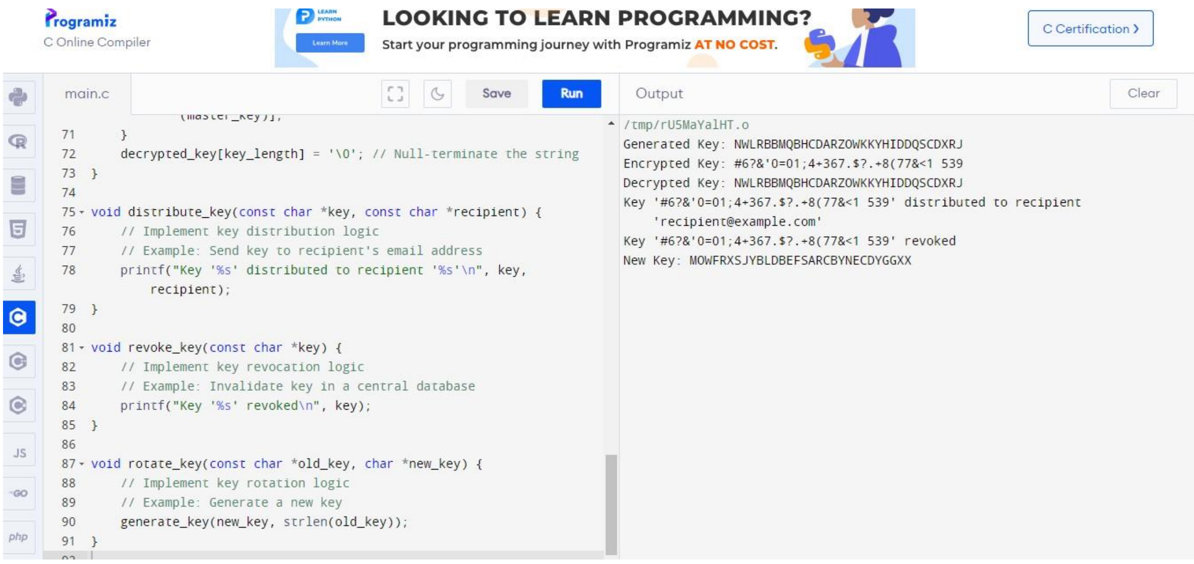
PROGRAM

```
#include  <stdio.h>

#include <stdlib.h>

#include <string.h>


// Function prototypes

void generate_key(char *key, size_t key_length);

void encrypt_key(const char *master_key, const char *key_to_encrypt, char *encrypted_key);

void decrypt_key(const char *master_key, const char *encrypted_key, char *decrypted_key);

void distribute_key(const char *key, const char *recipient);

void revoke_key(const char *key);

void rotate_key(const char *old_key, char *new_key);


int main() {

   // Example usage of key management functions

   char master_key[] = "masterpassword";

   char key[32];

   char encrypted_key[64];

   char decrypted_key[32];

   char new_key[32];


   // Generate a new key
```

```c
    generate_key(key, sizeof(key));
    printf("Generated Key: %s\n", key);

    // Encrypt the key using the master key
    encrypt_key(master_key, key, encrypted_key);
    printf("Encrypted Key: %s\n", encrypted_key);

    // Decrypt the key using the master key
    decrypt_key(master_key, encrypted_key, decrypted_key);
    printf("Decrypted Key: %s\n", decrypted_key);

    // Distribute the key to a recipient
    distribute_key(encrypted_key, "recipient@example.com");

    // Revoke the key
    revoke_key(encrypted_key);

    // Rotate the key
    rotate_key(key, new_key);
    printf("New Key: %s\n", new_key);

    return 0;
}

void generate_key(char *key, size_t key_length) {
    // Implement key generation logic
    // Example: Generate a random key
```

```
    for (size_t i = 0; i < key_length - 1; ++i) {
        key[i] = 'A' + rand() % 26; // Example: Random ASCII character
    }
    key[key_length - 1] = '\0'; // Null-terminate the string
}

void encrypt_key(const char *master_key, const char *key_to_encrypt, char *encrypted_key) {
    // Implement key encryption logic using the master key
    // Example: Simple XOR encryption
    size_t key_length = strlen(key_to_encrypt);
    for (size_t i = 0; i < key_length; ++i) {
        encrypted_key[i] = key_to_encrypt[i] ^ master_key[i % strlen(master_key)];
    }
    encrypted_key[key_length] = '\0'; // Null-terminate the string
}

void decrypt_key(const char *master_key, const char *encrypted_key, char *decrypted_key) {
    // Implement key decryption logic using the master key
    // Example: Simple XOR decryption
    size_t key_length = strlen(encrypted_key);
    for (size_t i = 0; i < key_length; ++i) {
        decrypted_key[i] = encrypted_key[i] ^ master_key[i % strlen(master_key)];
    }
    decrypted_key[key_length] = '\0'; // Null-terminate the string
}

void distribute_key(const char *key, const char *recipient) {
```

```
    // Implement key distribution logic
    // Example: Send key to recipient's email address
    printf("Key '%s' distributed to recipient '%s'\n", key, recipient);
}

void revoke_key(const char *key) {
    // Implement key revocation logic
    // Example: Invalidate key in a central database
    printf("Key '%s' revoked\n", key);
}

void rotate_key(const char *old_key, char *new_key) {
    // Implement key rotation logic
    // Example: Generate a new key
    generate_key(new_key, strlen(old_key));
}
```



PROGRAM -4

4	ECC	Implement a program that performs key generation, encryption, and decryption using elliptic curve cryptography.	1.30 – 2.00 PM

100 points Due Yesterday, 3:00 PM

		Compare the security and efficiency of ECC with traditional public-key cryptosystems like RSA	

ALGORITHM:

1. Take the message and the key (an integer) as input.
2. Declare a character array to store the message and read the message from the user and store it in the array.
3. Iterate through each character in the message.
4. For each alphabetic character (a-z or A-Z), shift it by the key value.
5. Print the encrypted message.
6. Take the encrypted message and the key as input.
7. Declare a character array to store the decrypted message and read the encrypted message from the user and store it in the array.
8. Iterate through each character in the encrypted message.
9. For each alphabetic character (a-z or A-Z), shift it back by the key value.
10. Print the decrypted message.

PROGRAM

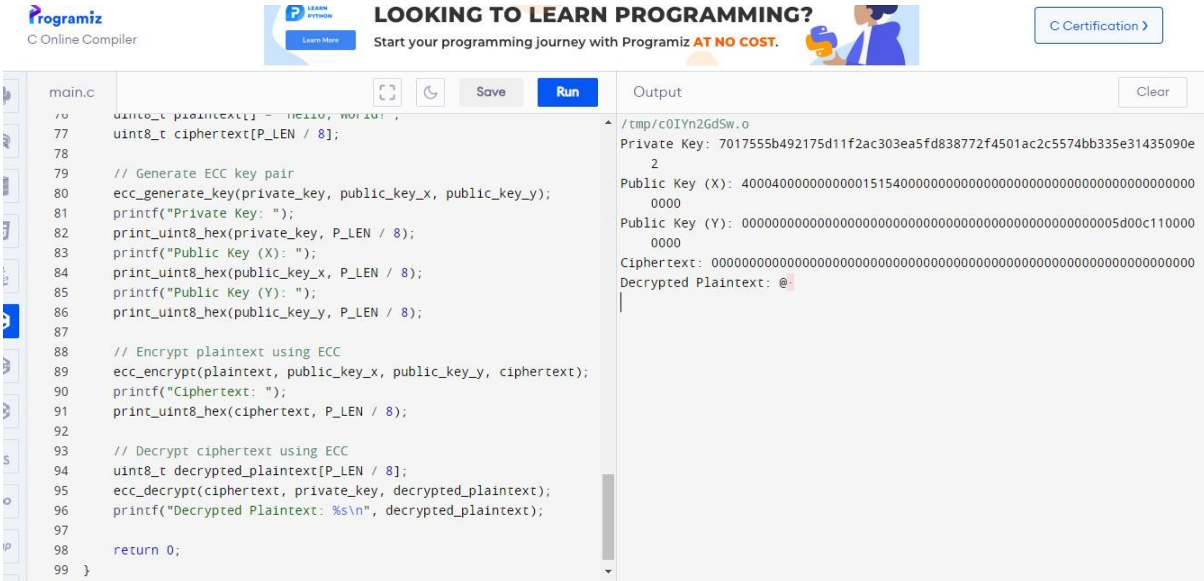
```
#include <stdio.h>

#include <stdint.h>

#include <stdlib.h>

#include <string.h>


// Elliptic Curve Parameters (Example: secp256k1)

#define P_LEN 256

#define PRIME_STR "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F"

#define A_STR "0000000000000000000000000000000000000000000000000000000000000000"

#define B_STR "0000000000000000000000000000000000000000000000000000000000000007"

#define GX_STR "79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798"
```

```c
#define GY_STR "483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8"
#define N_STR "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141"

// Helper Functions
void hex_string_to_uint8(const char *hex_str, uint8_t *result, size_t len) {
    for (size_t i = 0; i < len; i++) {
        sscanf(hex_str + 2 * i, "%2hhx", &result[i]);
    }
}

void print_uint8_hex(uint8_t *data, size_t len) {
    for (size_t i = 0; i < len; i++) {
        printf("%02x", data[i]);
    }
    printf("\n");
}

void ecc_point_add(uint8_t *x1, uint8_t *y1, uint8_t *x2, uint8_t *y2, uint8_t *x3, uint8_t *y3) {
    // Implement point addition algorithm for ECC
    // ...
}

void ecc_point_multiply(uint8_t *x, uint8_t *y, uint8_t *d, uint8_t *x_res, uint8_t *y_res) {
    // Implement point multiplication algorithm for ECC
    // ...
}
```

```
// ECC Key Generation
void ecc_generate_key(uint8_t *private_key, uint8_t *public_key_x, uint8_t *public_key_y) {
    // Initialize elliptic curve parameters
    uint8_t p[P_LEN / 8], a[P_LEN / 8], b[P_LEN / 8], gx[P_LEN / 8], gy[P_LEN / 8], n[P_LEN / 8];
    hex_string_to_uint8(PRIME_STR, p, P_LEN / 8);
    hex_string_to_uint8(A_STR, a, P_LEN / 8);
    hex_string_to_uint8(B_STR, b, P_LEN / 8);
    hex_string_to_uint8(GX_STR, gx, P_LEN / 8);
    hex_string_to_uint8(GY_STR, gy, P_LEN / 8);
    hex_string_to_uint8(N_STR, n, P_LEN / 8);

    // Generate a random private key
    // Example: Generate a 32-byte random private key
    // Use a cryptographically secure random number generator for production
    FILE *fp = fopen("/dev/urandom", "r");
    fread(private_key, 1, P_LEN / 8, fp);
    fclose(fp);

    // Compute public key using point multiplication
    ecc_point_multiply(gx, gy, private_key, public_key_x, public_key_y);
}

// ECC Encryption and Decryption
void ecc_encrypt(uint8_t *plaintext, uint8_t *public_key_x, uint8_t *public_key_y, uint8_t
*ciphertext) {
    // Encrypt plaintext using ECC
    // Example: Compute shared secret point and use it to derive a symmetric key for AES encryption
    // ...
```

```
}

void ecc_decrypt(uint8_t *ciphertext, uint8_t *private_key, uint8_t *plaintext) {
    // Decrypt ciphertext using ECC
    // Example: Compute shared secret point and use it to derive a symmetric key for AES decryption
    // ...
}

int main() {
    uint8_t private_key[P_LEN / 8], public_key_x[P_LEN / 8], public_key_y[P_LEN / 8];
    uint8_t plaintext[] = "Hello, world!";
    uint8_t ciphertext[P_LEN / 8];

    // Generate ECC key pair
    ecc_generate_key(private_key, public_key_x, public_key_y);
    printf("Private Key: ");
    print_uint8_hex(private_key, P_LEN / 8);
    printf("Public Key (X): ");
    print_uint8_hex(public_key_x, P_LEN / 8);
    printf("Public Key (Y): ");
    print_uint8_hex(public_key_y, P_LEN / 8);

    // Encrypt plaintext using ECC
    ecc_encrypt(plaintext, public_key_x, public_key_y, ciphertext);
    printf("Ciphertext: ");
    print_uint8_hex(ciphertext, P_LEN / 8);
```

```
// Decrypt ciphertext using ECC

uint8_t decrypted_plaintext[P_LEN / 8];

ecc_decrypt(ciphertext, private_key, decrypted_plaintext);

printf("Decrypted Plaintext: %s\n", decrypted_plaintext);


return 0;

}
```



5. PROGRAM -5

5	RC4	Develop a program to encrypt and decrypt messages using a stream cipher like the RC4 algorithm. Investigate the properties of stream ciphers compared to block ciphers. What are the advantages and disadvantages of each?	2.00 – 3.00 PM

ALGORITHM:

1. Take the message and the key (an integer) as input.
2. Declare a character array to store the message and read the message from the user and store it in the array.
3. Iterate through each character in the message.
4. For each alphabetic character (a-z or A-Z), shift it by the key value.
5. Print the encrypted message.
6. Take the encrypted message and the key as input.
7. Declare a character array to store the decrypted message and read the encrypted message from the user and store it in the array.
8. Iterate through each character in the encrypted message.

9. For each alphabetic character (a-z or A-Z), shift it back by the key value.
10. Print the decrypted message.

PROGRAM

```
#include <stdio.h>

#include <stdint.h>

#include <string.h>


// RC4 State

typedef struct {

   uint8_t s[256];

   uint8_t i;

   uint8_t j;

} RC4State;


// RC4 Key Setup

void rc4_key_setup(RC4State *state, const uint8_t *key, size_t key_length) {

   // Initialize state array

   for (int i = 0; i < 256; i++) {

      state->s[i] = i;

   }


   // Permute state array based on the key

   uint8_t j = 0;

   for (int i = 0; i < 256; i++) {

      j = (j + state->s[i] + key[i % key_length]) % 256;

      uint8_t temp = state->s[i];

      state->s[i] = state->s[j];

      state->s[j] = temp;
```

```
    }

    // Reset indices
    state->i = 0;
    state->j = 0;
}

// RC4 Pseudorandom Generation Algorithm (PRGA)
uint8_t rc4_prga(RC4State *state) {
    state->i = (state->i + 1) % 256;
    state->j = (state->j + state->s[state->i]) % 256;

    // Swap s[i] and s[j]
    uint8_t temp = state->s[state->i];
    state->s[state->i] = state->s[state->j];
    state->s[state->j] = temp;

    return state->s[(state->s[state->i] + state->s[state->j]) % 256];
}

// RC4 Encrypt or Decrypt Message
void rc4_crypt(const uint8_t *input, size_t input_length, const uint8_t *key, size_t key_length,
uint8_t *output) {
    RC4State state;
    rc4_key_setup(&state, key, key_length);

    for (size_t k = 0; k < input_length; k++) {
        output[k] = input[k] ^ rc4_prga(&state);
```

```c
    }
}

int main() {
    // Example key and message
    const uint8_t key[] = "SecretKey";
    const uint8_t plaintext[] = "Hello, world!";
    const size_t plaintext_length = strlen((char *)plaintext);

    // Buffer for ciphertext
    uint8_t ciphertext[plaintext_length];

    // Encrypt plaintext
    rc4_crypt(plaintext, plaintext_length, key, strlen((char *)key), ciphertext);

    // Print ciphertext
    printf("Ciphertext: ");
    for (size_t i = 0; i < plaintext_length; i++) {
        printf("%02X ", ciphertext[i]);
    }
    printf("\n");

    // Decrypt ciphertext
    uint8_t decrypted_plaintext[plaintext_length];
    rc4_crypt(ciphertext, plaintext_length, key, strlen((char *)key), decrypted_plaintext);

    // Print decrypted plaintext
```

```
    printf("Decrypted Plaintext: %s\n", decrypted_plaintext);


    return 0;
}
```