```
      }
    }
}

int main() {
    char ciphertext[1000];
    char key[100];

    printf("Enter ciphertext: ");
    fgets(ciphertext, sizeof(ciphertext), stdin);
    ciphertext[strcspn(ciphertext, "\n")] = 0;

    printf("Enter key: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = 0;

    vigenere_decrypt(ciphertext, key);
    printf("Decrypted text: %s\n", ciphertext);

    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/
" && gcc Prac_5_2.c -o Prac_5_2 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Pr
ac/"Prac_5_2
Enter ciphertext: omkatp ftyjrt
Enter key: test
Decrypted text: vishal maurya
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

Practical-6

Aim: A) Transposition Encryption

Theory:

The Rail Fence Cipher is a transposition cipher where the plaintext is written in a zigzag pattern across multiple "rails" (rows). To encrypt, the message is written diagonally across the chosen number of rails, then read row by row to form the ciphertext. Decryption reverses this by reconstructing the zigzag pattern and reading it to retrieve the original message. This method is simple and relies on rearranging the characters rather than altering them.

Code:

```
#include <stdio.h>
#include <string.h>

void rail_fence_encrypt(const char *plaintext, int num_rails, char *ciphertext) {
   if (num_rails <= 1) {
      strcpy(ciphertext, plaintext);
      return;
   }
   int len = strlen(plaintext);
   char rails[num_rails][len];

   for (int i = 0; i < num_rails; i++) {
      for (int j = 0; j < len; j++) {
         rails[i][j] = '\0';
      }
   }
   int rail_direction = 1;
   int current_rail = 0;

   for (int i = 0; i < len; i++) {
      rails[current_rail][i] = plaintext[i];
```


HARDIK GODKAR
210303105029
7B23

```
        current_rail += rail_direction;
        if (current_rail == 0 || current_rail == num_rails - 1) {
            rail_direction *= -1;
        }
    }
    int index = 0;
    for (int i = 0; i < num_rails; i++) {
        for (int j = 0; j < len; j++) {
            if (rails[i][j] != '\0') {
                ciphertext[index++] = rails[i][j];
            }
        }
    }
    ciphertext[index] = '\0';
}
int main() {
    const char plaintext[] = "vishal";
    int num_rails = 3;
    char ciphertext[strlen(plaintext) + 1];

    printf("Plaintext: %s\n", plaintext);
    rail_fence_encrypt(plaintext, num_rails, ciphertext);
    printf("Ciphertext: %s\n", ciphertext);

    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  P
rac/" && gcc Pra
c_6_1.c -o Prac_6_1 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"Prac
_6_1
Plaintext: vishal
Ciphertext: vaihls
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

Aim: B) Transposition Decryption

Code:

```
#include <stdio.h>
#include <string.h>

void rail_fence_decrypt(const char *ciphertext, int num_rails, char *plaintext) {
   if (num_rails <= 1) {
      strcpy(plaintext, ciphertext);
      return;
   }

   int len = strlen(ciphertext);
   int rail_length[num_rails];
   memset(rail_length, 0, sizeof(rail_length));
   int rail_direction = 1;
   int current_rail = 0;

   for (int i = 0; i < len; i++) {
      rail_length[current_rail]++;
      current_rail += rail_direction;
      if (current_rail == 0 || current_rail == num_rails - 1) {
         rail_direction *= -1;
```


HARDIK GODKAR
210303105029
7B23

```
    }
}

char rails[num_rails][len];
int index = 0;
for (int i = 0; i < num_rails; i++) {
    for (int j = 0; j < rail_length[i]; j++) {
        rails[i][j] = ciphertext[index++];
    }
}
rail_direction = 1;
current_rail = 0;
int rail_index[num_rails];
memset(rail_index, 0, sizeof(rail_index));

int plaintext_index = 0;
for (int i = 0; i < len; i++) {
    plaintext[plaintext_index++] = rails[current_rail][rail_index[current_rail]++];
    current_rail += rail_direction;
    if (current_rail == 0 || current_rail == num_rails - 1) {
        rail_direction *= -1;
    }
}
plaintext[plaintext_index] = '\0';
```


HARDIK GODKAR
210303105029
7B23

```
}
int main() {
   const char ciphertext[] = "vaihls";
   int num_rails = 3;
   char plaintext[strlen(ciphertext) + 1];
   printf("Ciphertext: %s\n", ciphertext);
   rail_fence_decrypt(ciphertext, num_rails, plaintext);
   printf("Plaintext: %s\n", plaintext);
   return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS
 Prac/" && gcc Prac_6_2.c -o Prac_6_2 && "/Users/vishalmaurya/Documents/Study/Sem 7
/INS/INS  Prac/"Prac_6_2
Ciphertext: vaihls
Plaintext: vishal
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

Practical-7

AIM: Implement Diffie- Hellman Key exchange Method.

Theory:

Steps for Diffie-Hellman Key Exchange

1. Agree on Public Parameters:

- o A prime number p
- o A base (or generator) g

2. Generate Private Keys:

- o Each party generates a private key.

3. Compute Public Keys:

- o Each party computes their public key using the formula:

public_key=g^private_key mod p

4. Exchange Public Keys:

- o Each party sends their public key to the other party.

5. Compute the Shared Secret:

- o Each party computes the shared secret using the received public key and their

own private key:

- o shared_secret= received_public_key^private_key mod p

Code:

```
#include <stdio.h>

int power(int a, int b, int p) {

   int result = 1;
```


HARDIK GODKAR
210303105029
7B23

```
    a = a % p;
    while (b > 0) {
        if (b % 2 == 1) {
            result = (result * a) % p;
        }
        b = b >> 1;
        a = (a * a) % p;
    }
    return result;
}

int main() {
    int P = 23;
    printf("The value of P: %d\n", P);
    int G = 9;
    printf("The value of G: %d\n", G);
    int a = 4;
    printf("The private key a for Alice: %d\n", a);
    int x = power(G, a, P);
    int b = 3;
    printf("The private key b for Bob: %d\n", b);
    int y = power(G, b, P);
    int ka = power(y, a, P);
    int kb = power(x, b, P);
```


HARDIK GODKAR
210303105029
7B23

```
    printf("Secret key for Alice is: %d\n", ka);
    printf("Secret key for Bob is: %d\n", kb);
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS
 Prac/" && gcc Prac_7.c -o Prac_7 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS
/INS  Prac/"Prac_7
The value of P: 23
The value of G: 9
The private key a for Alice: 4
The private key b for Bob: 3
Secret key for Alice is: 9
Secret key for Bob is: 9
```


HARDIK GODKAR
210303105029
7B23

PRACTICAL-8

AIM:(A) Implement One time pad encryption.

Theory:

1. Basic Principle: The key must be random, as long as the message, and used only once to produce ciphertext by combining it with the plaintext.

2. Encryption Process:: XOR is commonly used for binary data, and modular arithmetic for alphabetic characters to encrypt and decrypt.

3. Properties: OTP provides perfect secrecy with a truly random key used only once, ensuring symmetric encryption and key non-reusability.

4. Key Generation: The key must be as long as the plaintext, truly random, and ideally generated by hardware-based random number generators.

5. Security Considerations: The key must be truly random, securely distributed, and properly stored to prevent unauthorized decryption.

6. Vulnerabilities: Reusing the key or poor key management can lead to vulnerabilities like crib dragging attacks and practical difficulties.

7. Practical Use Cases: OTP is rarely used in modern cryptography but may be employed for highly secure, small-scale applications

HARDIK GODKAR
210303105029
7B23

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void generate_key(int length, char *key) {
    srand(time(0));
    for (int i = 0; i < length; i++) {
        key[i] = (rand() % 2) ? '1' : '0';
    }
    key[length] = '\0';
}

void string_to_binary(const char *text, char *binary) {
    while (*text) {
        unsigned char c = *text++;
        for (int i = 7; i >= 0; i--) {
            binary[i] = (c & 1) ? '1' : '0';
            c >>= 1;
        }
        binary += 8;
    }
    *binary = '\0';
```


HARDIK GODKAR
210303105029
7B23

```
}

void binary_to_string(const char *binary, char *text) {
    while (*binary) {
        unsigned char c = 0;
        for (int i = 0; i < 8; i++) {
            c = (c << 1) | (*binary++ - '0');
        }
        *text++ = c;
    }
    *text = '\0';
}

void xor_binary(const char *binary1, const char *binary2, char *result) {
    while (*binary1) {
        *result++ = (*binary1++ != *binary2++) ? '1' : '0';
    }
    *result = '\0';
}

void one_time_pad_encrypt(const char *plaintext, const char *key, char *ciphertext) {
    char binary_plaintext[1024];
    string_to_binary(plaintext, binary_plaintext);
    xor_binary(binary_plaintext, key, ciphertext);
```


HARDIK GODKAR
210303105029
7B23

```
}

void one_time_pad_decrypt(const char *ciphertext, const char *key, char *decrypted_text) {
    char decrypted_binary[1024];
    xor_binary(ciphertext, key, decrypted_binary);
    binary_to_string(decrypted_binary, decrypted_text);
}

int main() {
    const char *plaintext = "i am vishal";
    char binary_plaintext[1024];
    char key[1024];
    char ciphertext[1024];
    char decrypted_text[1024];

    string_to_binary(plaintext, binary_plaintext);
    generate_key(strlen(binary_plaintext), key);
    one_time_pad_encrypt(plaintext, key, ciphertext);

    printf("Plaintext: %s\n", plaintext);
    printf("Generated Key: %s\n", key);
    printf("Ciphertext (binary): %s\n", ciphertext);

    one_time_pad_decrypt(ciphertext, key, decrypted_text);
```


HARDIK GODKAR
210303105029
7B23

```
    printf("Decrypted Text: %s\n", decrypted_text);

    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/" &&
 gcc Prac_8_1.c -o Prac_8_1 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"Prac
_8_1
Plaintext: i am vishal
Generated Key: 1100011011101000010100101101001100010100110100011111101001010000001001001111
1101110111
Ciphertext (binary): 1010111111001000001100111011111000110100101001111001001100100011011110100
0001101000011011
Decrypted Text: i am vishal
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

AIM:(B) Implement One Time Pad Decryption.

Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void binary_to_text(const char *binary_str, char *result) {
    size_t length = strlen(binary_str);
    for (size_t i = 0; i < length; i += 8) {
        char byte[9];
        strncpy(byte, binary_str + i, 8);
        byte[8] = '\0';
        unsigned char character = (unsigned char) strtol(byte, NULL, 2);
        *result++ = character;
    }
    *result = '\0';
}
void otp_decrypt(const char *ciphertext, const char *key, char *plaintext) {
    if (strlen(ciphertext) != strlen(key)) {
        fprintf(stderr, "Ciphertext and key must be of the same length.\n");
        exit(EXIT_FAILURE);
    }
    char plaintext_binary[1024];
    size_t length = strlen(ciphertext);
    for (size_t i = 0; i < length; ++i) {
```


HARDIK GODKAR
210303105029
7B23

```
        plaintext_binary[i] = (ciphertext[i] != key[i]) ? '1' : '0';
    }
    plaintext_binary[length] = '\0';
    binary_to_text(plaintext_binary, plaintext);
}
int main() {
    const char *ciphertext =
"10101111110010000011001110111110001101001010011110010011001000110111101000011010000011011";
  const char *key =
"11000110111010000101001011010011000101001101000111111010010100000000100100111101101110111";
    char plaintext[1024];
    otp_decrypt(ciphertext, key, plaintext);
    printf("Decrypted plaintext: %s\n", plaintext);
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 && gcc Prac_8_2.c -o Prac_8_2 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 Prac_8_2
 Decrypted plaintext: i am vishal
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

PRACTICAL-9

AIM:(A) Implement RSA encryption algorithm.

Theory:

RSA is an asymmetric encryption algorithm that uses a pair of keys: a public key for encryption and a private key for decryption. The key generation process involves selecting two large prime numbers, computing their product to form the modulus, and choosing a public exponent along with a private exponent that ensures specific mathematical properties. Encryption is done by applying an exponentiation operation to the message using the public key, resulting in ciphertext. The security of RSA relies on the difficulty of factoring the product of the two primes. While encryption is generally efficient, decryption is slower but can be optimized. RSA is used in various applications, including digital signatures, secure communications in protocols like TLS/SSL, and Public Key Infrastructure (PKI) for managing digital certificates.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
int gcd(int a, int b) {
   while (b != 0) {
      int temp = b;
      b = a % b;
      a = temp;
   }
```


HARDIK GODKAR
210303105029
7B23

```
    return a;
}
int modinv(int e, int phi) {
    int d = 0, x1 = 0, x2 = 1, y1 = 1;
    int temp_phi = phi;
    while (e > 0) {
        int temp1 = temp_phi / e;
        int temp2 = temp_phi - temp1 * e;
        temp_phi = e;
        e = temp2;
        int x = x2 - temp1 * x1;
        int y = d - temp1 * y1;
        x2 = x1;
        x1 = x;
        d = y1;
        y1 = y;
    }
    if (temp_phi == 1) {
        return d + phi;
    }
    return d;
}
void generate_keypair(int p, int q, int *e, int *d, int *n) {
    *n = p * q;
```


HARDIK GODKAR
210303105029
7B23

```
    int phi = (p - 1) * (q - 1);
    srand(time(0));
    *e = rand() % (phi - 1) + 1;
    while (gcd(*e, phi) != 1) {
        *e = rand() % (phi - 1) + 1;
    }
    *d = modinv(*e, phi);
}
void encrypt(int key, int n, const char *plaintext, int *cipher, int *cipher_len) {
    int i = 0;
    while (plaintext[i] != '\0') {
        cipher[i] = (int)pow((int)plaintext[i], key) % n;
        i++;
    }
    *cipher_len = i;
}
int main() {
    int p = 61;
    int q = 53;
    int e, d, n;
    generate_keypair(p, q, &e, &d, &n);
    printf("Public key: (%d, %d)\n", e, n);
    printf("Private key: (%d, %d)\n", d, n);
    const char *message = "HELLO VISHAL";
```


HARDIK GODKAR
210303105029
7B23

```
    int encrypted_message[256];
    int cipher_len;
    encrypt(e, n, message, encrypted_message, &cipher_len);
    printf("Original message: %s\n", message);
    printf("Encrypted message: ");
    for (int i = 0; i < cipher_len; i++) {
        printf("%d ", encrypted_message[i]);
    }
    printf("\n");
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 && gcc Prac_9_1.c -o Prac_9_1 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 Prac_9_1
 Public key: (769, 3233)
 Private key: (4609, 3233)
 Original message: HELLO VISHAL
 Encrypted message: 2193 2193 2193 2193 2193 2193 2193 2193 2193 2193 2193 2193
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

AIM(B): Implement RSA decryption algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
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 >> 1;
      base = (base * base) % mod;
   }
   return result;
}
int gcd(int a, int b) {
   while (b != 0) {
      int temp = b;
      b = a % b;
      a = temp;
   }
   return a;
}
```


HARDIK GODKAR
210303105029
7B23

```
int modinv(int e, int phi) {
    int d = 0, x1 = 0, x2 = 1, y1 = 1;
    int temp_phi = phi;
    while (e > 0) {
        int temp1 = temp_phi / e;
        int temp2 = temp_phi - temp1 * e;
        temp_phi = e;
        e = temp2;
        int x = x2 - temp1 * x1;
        int y = d - temp1 * y1;
        x2 = x1;
        x1 = x;
        d = y1;
        y1 = y;
    }
    if (temp_phi == 1) {
        return d + phi;
    }
    return d;
}
void generate_keypair(int p, int q, int* e, int* d, int* n) {
    int phi = (p - 1) * (q - 1);
    *n = p * q;
```


HARDIK GODKAR
210303105029
7B23

```
    srand(time(0));
    *e = rand() % (phi - 1) + 1;
    while (gcd(*e, phi) != 1) {
        *e = rand() % (phi - 1) + 1;
    }
    *d = modinv(*e, phi);
}
void encrypt(int e, int n, const char* plaintext, int* ciphertext, int len) {
    for (int i = 0; i < len; i++) {
        ciphertext[i] = mod_exp(plaintext[i], e, n);
    }
}
void decrypt(int d, int n, int* ciphertext, char* plaintext, int len) {
    for (int i = 0; i < len; i++) {
        plaintext[i] = (char)mod_exp(ciphertext[i], d, n);
    }
    plaintext[len] = '\0';
}
int main() {
    int p = 61;
    int q = 53;
    int e, d, n;
    generate_keypair(p, q, &e, &d, &n);
    printf("Public key: (%d, %d)\n", e, n);
```


HARDIK GODKAR
210303105029
7B23

```
    printf("Private key: (%d, %d)\n", d, n);
    char message[] = "HELLO VISHAL";
    int len = sizeof(message) - 1;
    int encrypted_message[len];
    char decrypted_message[len + 1];
    encrypt(e, n, message, encrypted_message, len);
    printf("Encrypted message: ");
    for (int i = 0; i < len; i++) {
        printf("%d ", encrypted_message[i]);
    }
    printf("\n");
    decrypt(d, n, encrypted_message, decrypted_message, len);
    printf("Decrypted message: %s\n", decrypted_message);
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 && gcc Prac_9_2.c -o Prac_9_2 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 Prac_9_2
 Public key: (67, 3233)
 Private key: (1723, 3233)
 Encrypted message: 3039 521 83 83 1889 1676 561 1506 1825 3039 768 83
 Decrypted message: HELLO VISHAL
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

PRACTICAL-10

AIM: Demonstrate working of digital signature using cryptool.

Theory:

1. Overview of Digital Signatures

- **Purpose**: Digital signatures provide authenticity, integrity, and non-repudiation to digital communications and documents.
- **Components**: A digital signature process involves a key pair (public and private keys) and a hashing algorithm.

2. Key Concepts

- **Public and Private Keys**:
  - **Private Key**: Used to generate the signature.
  - **Public Key**: Used to verify the signature.
- **Hash Function**: Converts the original data into a fixed-size hash value. Common hash functions include SHA-256 and SHA-512.
- **Signature Generation**: The hash of the message is encrypted with the sender's private key to create a signature.
- **Signature Verification**: The recipient decrypts the signature with the sender's public key to retrieve the hash, then hashes the received message and compares it with the decrypted hash.

3. Steps to Demonstrate Digital Signatures Using CrypTool

1. **Generate Key Pair**:
   - **Generate a Public-Private Key Pair**:
     - Open CrypTool and select the digital signature or cryptographic module.
     - Generate a key pair using a cryptographic algorithm such as RSA or ECDSA.
     - Save the private key securely and the public key for verification purposes.
2. Create and Sign a Document:
   - **Prepare the Document**:
     - Write or load the document or message you want to sign.
   - Hash the Document:
     - CrypTool will automatically hash the document using a chosen hash algorithm (e.g., SHA-256).

HARDIK GODKAR
210303105029
7B23

3. Verify the Signature:
    - **Load the Document and Signature**:
        - Open CrypTool and load the document and its attached signature.
    - Hash the Document:
        - Compute the hash of the received document using the same hash function.
    - Decrypt the Signature:
        - Use the sender's public key to decrypt the signature and retrieve the original hash.
    - Compare Hashes:
        - Compare the computed hash with the decrypted hash. If they match, the signature is valid, confirming the document's integrity and authenticity.

4. Key Properties

- **Authenticity**: The digital signature verifies that the document was signed by the claimed sender.
- **Integrity**: Ensures that the document has not been altered after signing.
- **Non-repudiation**: The sender cannot deny the authenticity of the signature.

5. Use Cases

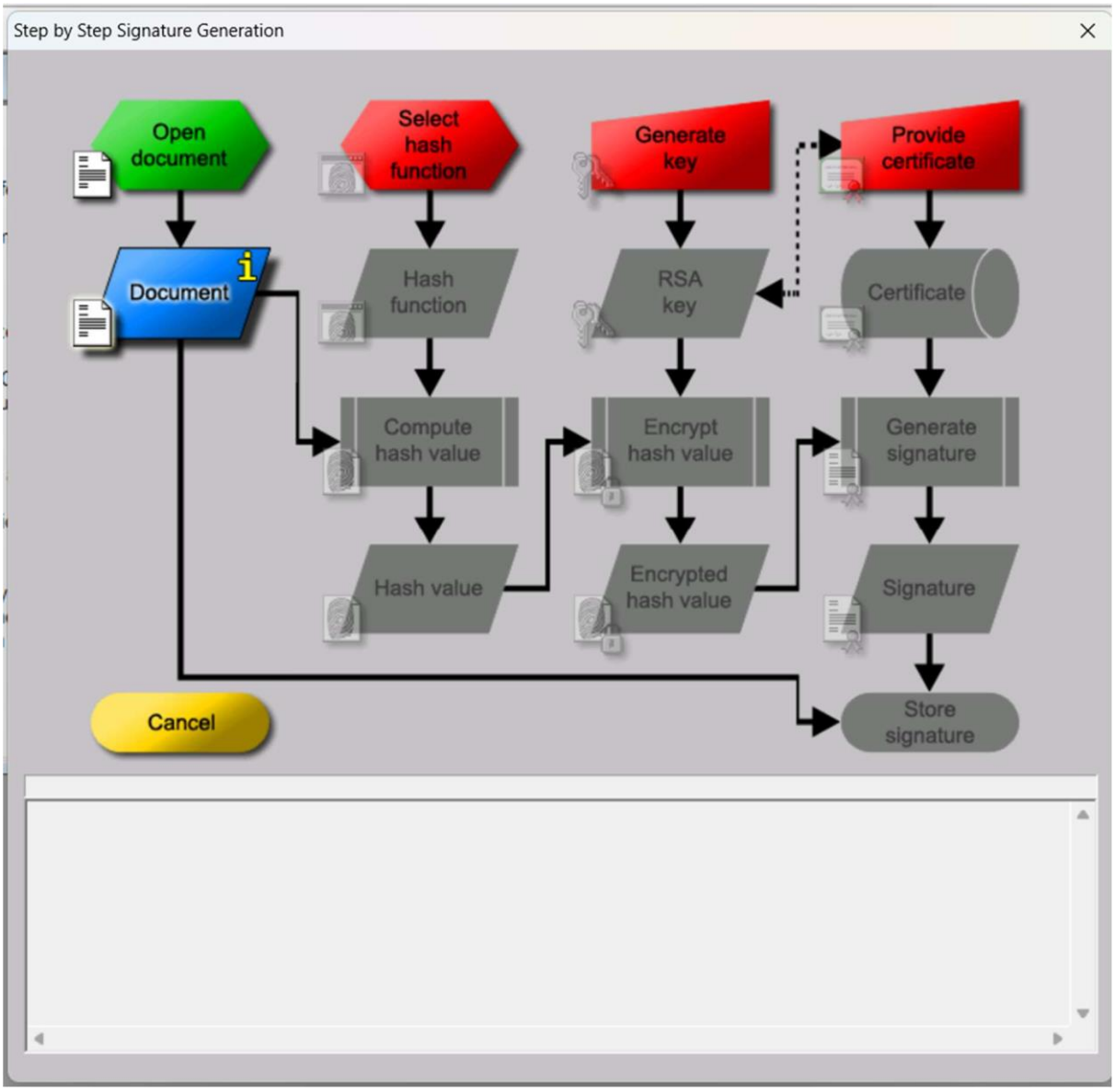
- **Email**: Digitally sign emails to confirm the sender's identity and ensure that the message hasn't been tampered with.
- **Software Distribution**: Sign software to verify the source and integrity of the software package.
- **Legal Documents**: Digitally sign contracts and legal documents for secure and verifiable transactions.

6. Security Considerations

- **Key Management**: Ensure private keys are kept secure and confidential.
- **Hash Function Choice**: Use strong, collision-resistant hash functions to prevent hash collisions.
- **Algorithm Choice**: Choose robust cryptographic algorithms with proven security properties.

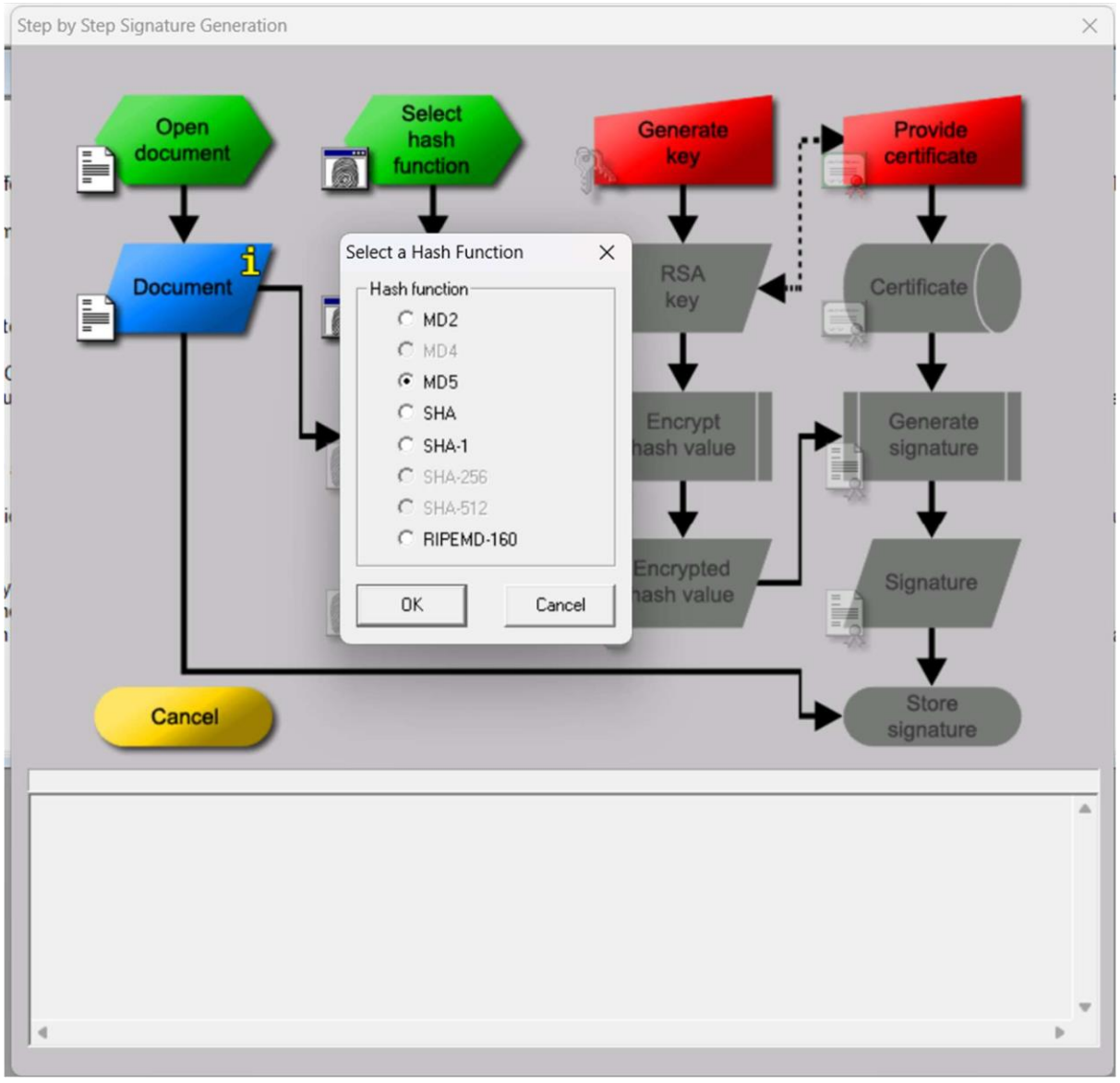
HARDIK GODKAR
210303105029
7B23

1.Open a document name file_name.txt.



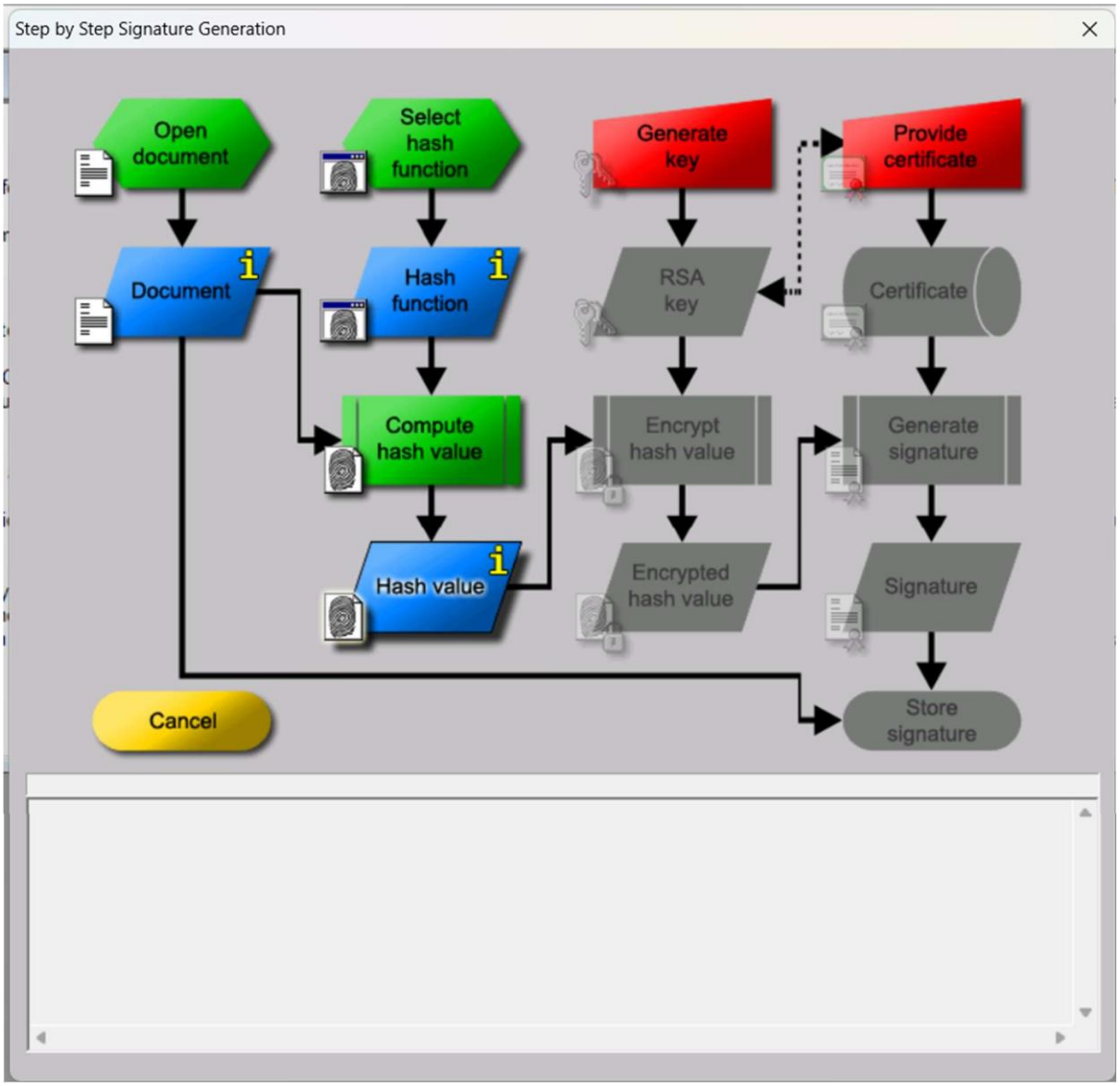
HARDIK GODKAR
210303105029
7B23

2.Select a Hash function.



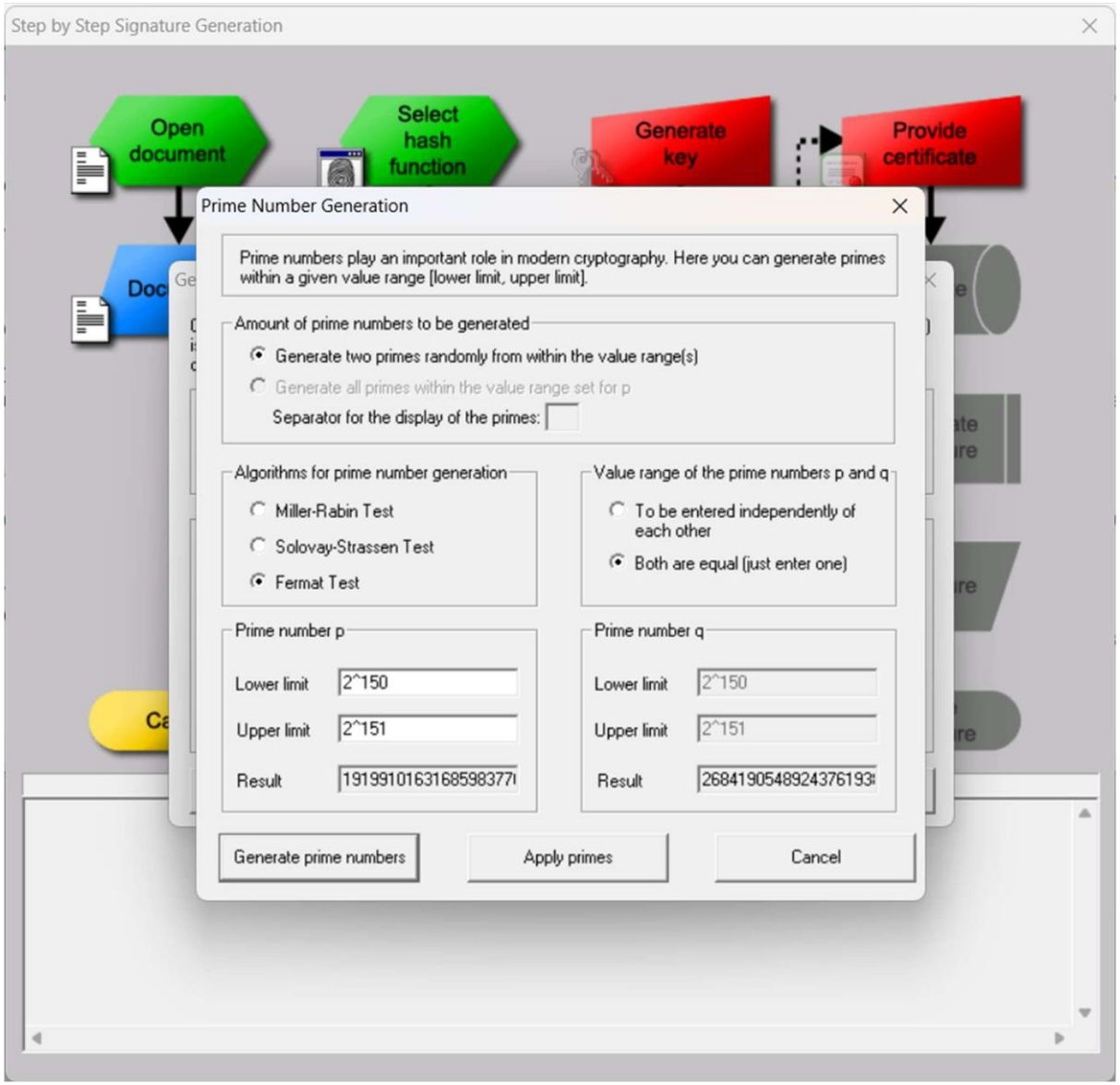
HARDIK GODKAR
210303105029
7B23

3.Compute Hash Value.



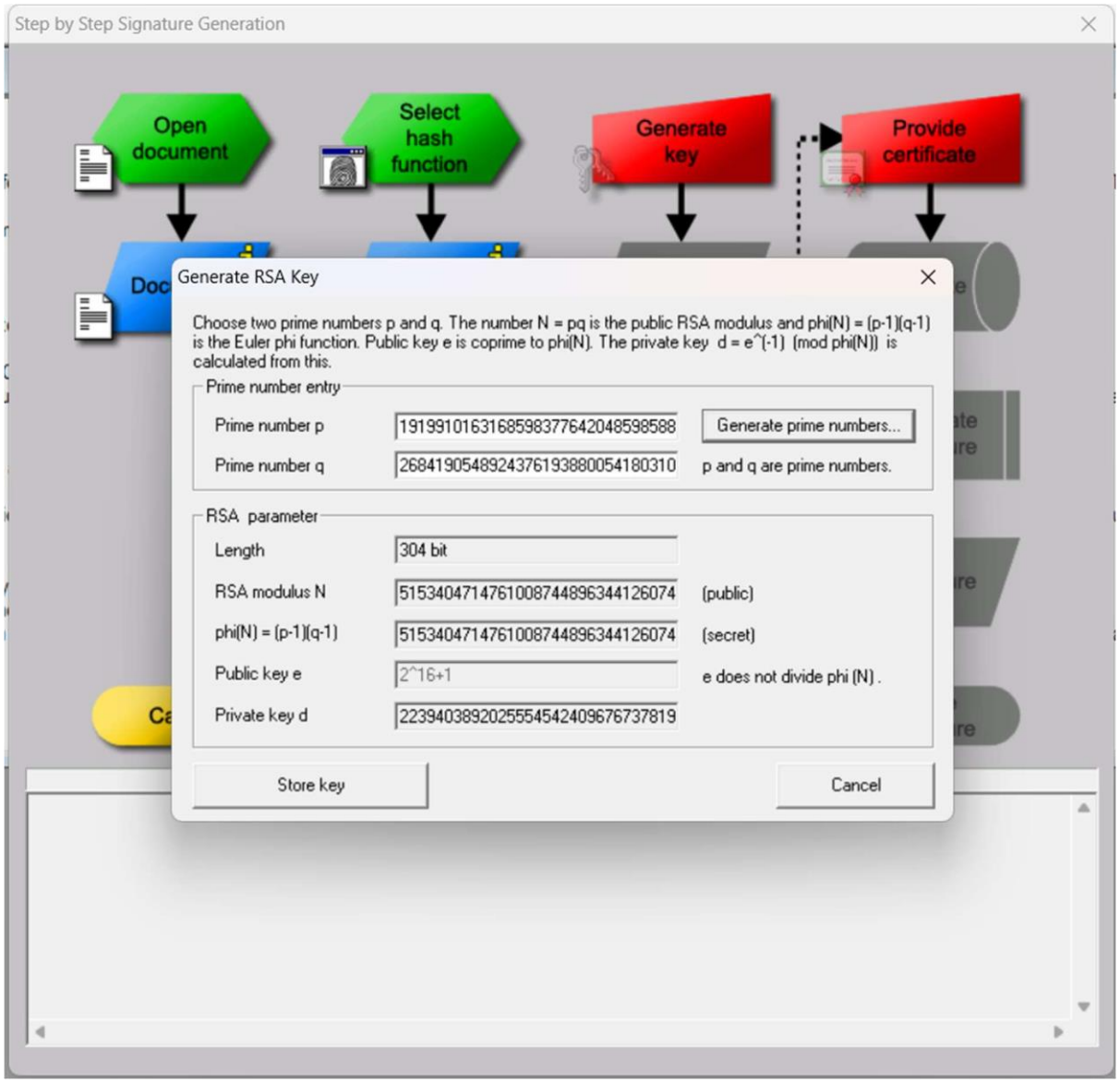
HARDIK GODKAR
210303105029
7B23

4. Generate key.



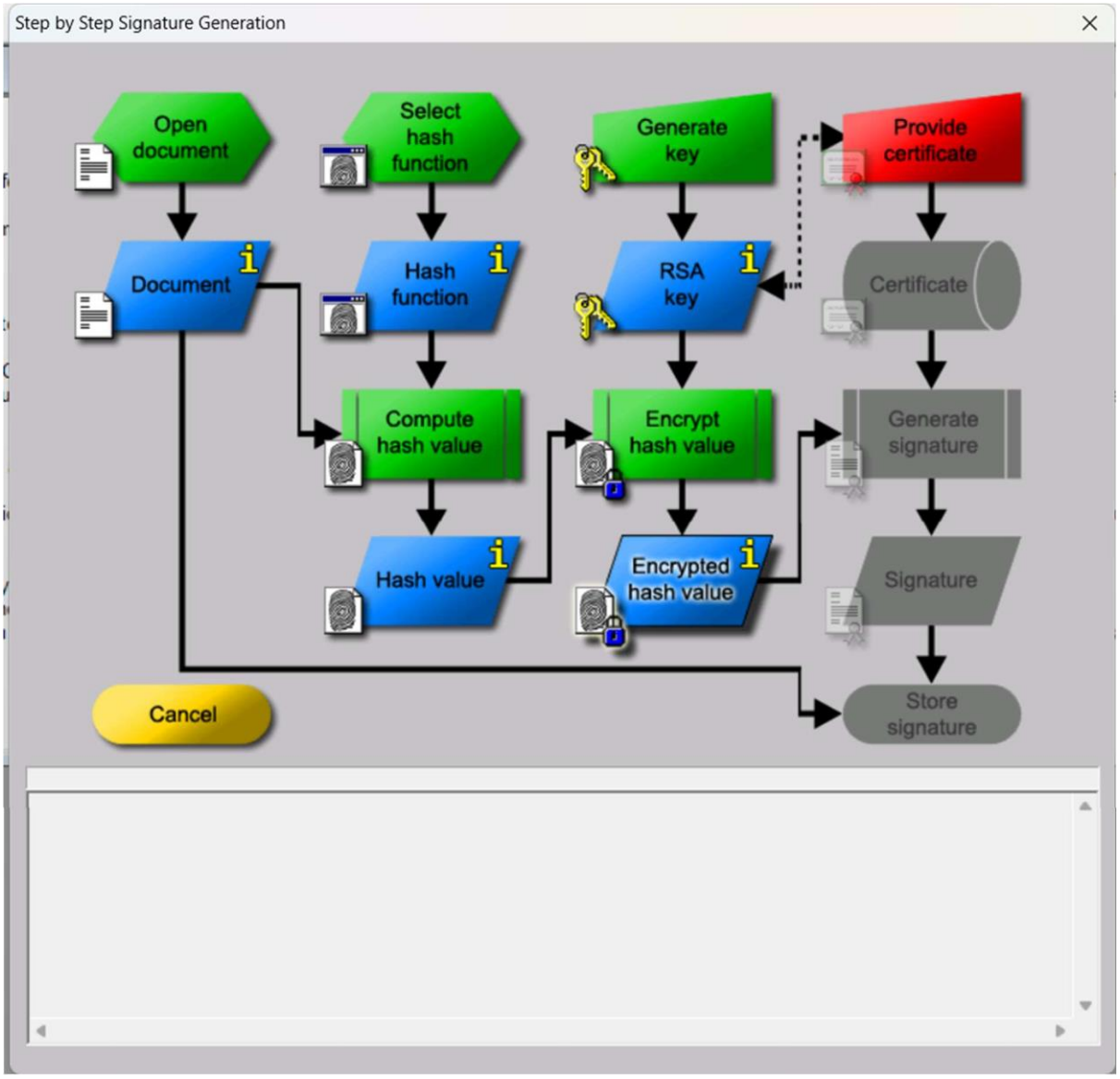
HARDIK GODKAR
210303105029
7B23

5. Store Keys.



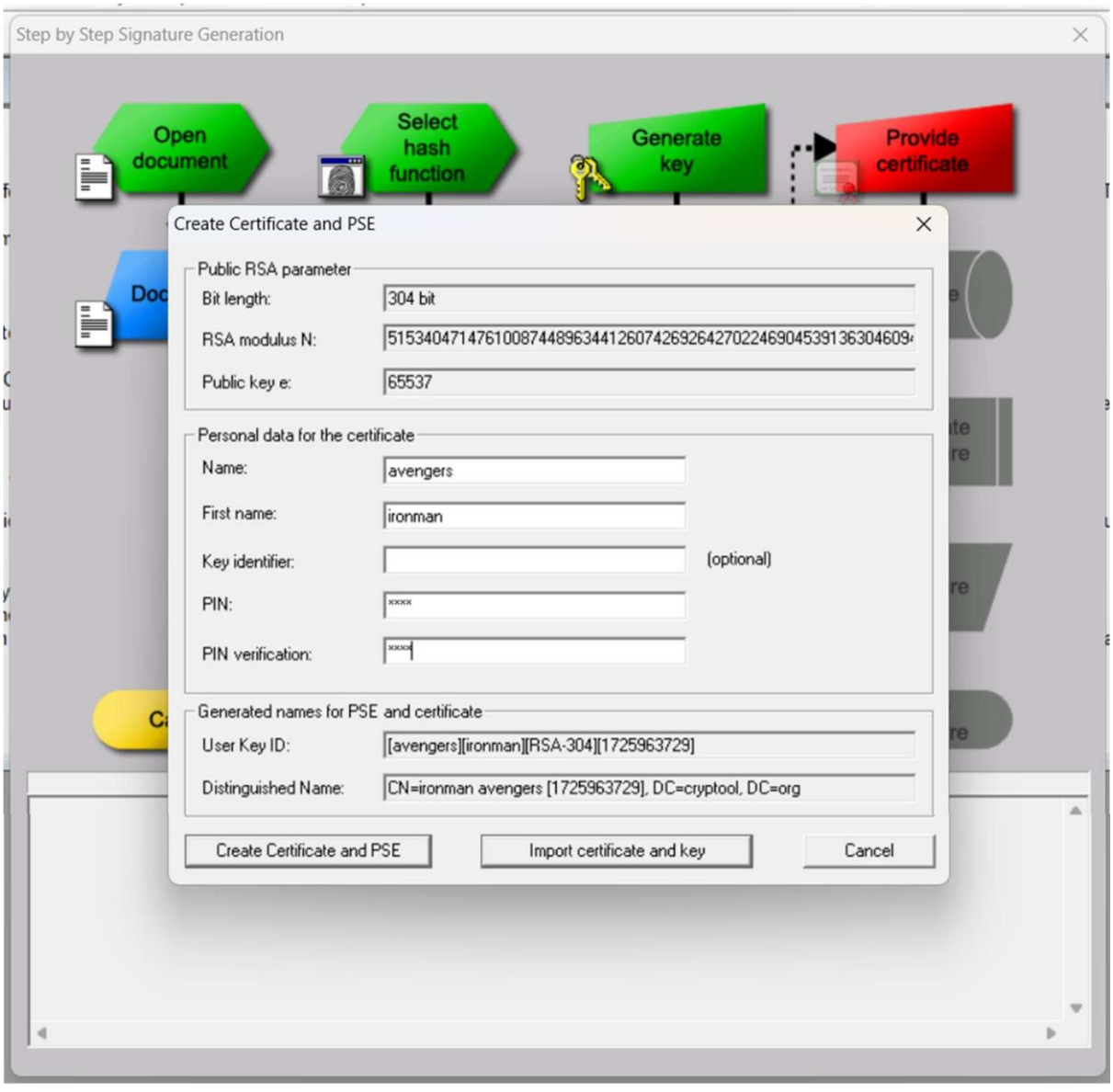
HARDIK GODKAR
210303105029
7B23

6.Encrypt Hash Value



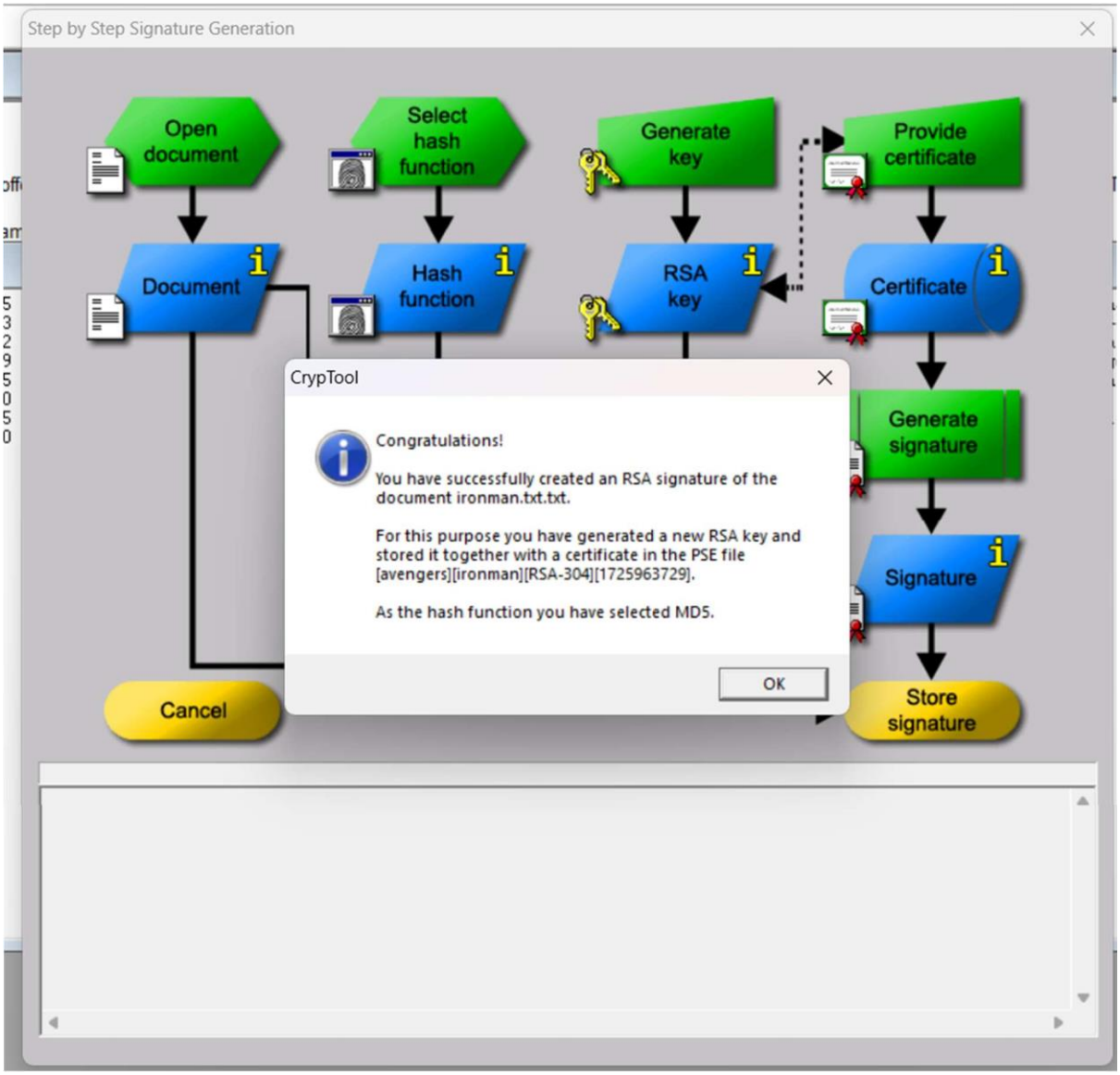
HARDIK GODKAR
210303105029
7B23

7.Provide certificate.



HARDIK GODKAR
210303105029
7B23

8.Store Signature.



Output:

HARDIK GODKAR
210303105029
7B23

HARDIK GODKAR
210303105029
7B23

Practical – 1

Aim:- (A) Caesar Cipher Encryption

Theory:-

• Caesar cipher is a substitution cipher where each letter in the plaintext is shifted a certain number of places down or up the alphabet.
• It's a simple and historical method of encryption, but it's not very secure due to its vulnerability to brute force attacks.
• The key idea is shifting letters by a fixed amount to encode and decode messages.
• For Caesar cipher encryption, each letter in the plaintext is shifted a fixed number of places down or up the alphabet.
• Let's say we have a shift of 3. So, 'A' becomes 'D', 'B' becomes 'E', and so on.
• To decrypt, you simply shift back by the same number of places.
• It's a straightforward method but not very secure.
• The key is crucial for both encryption and decryption.

Code:

```
#include <iostream>
#include <string>
using namespace std;



string encryption(string text, int shift) {
   string result = "";


   for (int i = 0; i < text.length(); i++) {
      char c = text[i];
```


HARDIK GODKAR
210303105029
7B23

```
        if (isupper(c)) {

            result += char((int(c - 'A') + shift) % 26 + 'A');
        } else if (islower(c)) {

            result += char((int(c - 'a') + shift) % 26 + 'a');
        } else {

            result += c;
        }
    }

    return result;
}

int main() {
    string text;
    int shift;

    cout << "Enter text to be encrypted: ";
    getline(cin, text);

    cout << "Enter shift value: ";
    cin >> shift;

    string encryptedText = encryption(text, shift);
    cout << "Encrypted Text: " << encryptedText << endl;

    return 0;
}
```


HARDIK GODKAR
210303105029
7B23

Output:

HARDIK GODKAR
210303105029
7B23

Aim:- (B) Caesar Cipher Decryption

Code:

```
#include <iostream>
#include <string>
using namespace std;

string decryption(string text, int shift) {
   string result = "";

   for (int i = 0; i < text.length(); i++) {
      char c = text[i];

      if (isupper(c)) {
         result += char((int(c - 'A' - shift + 26) % 26 + 'A'));
      } else if (islower(c)) {
         result += char((int(c - 'a' - shift + 26) % 26 + 'a'));
      } else {
         result += c;
      }
   }

   return result;
}

int main() {
   string text;
   int shift;

   cout << "Enter text to be decrypted: ";
```


HARDIK GODKAR
210303105029
7B23

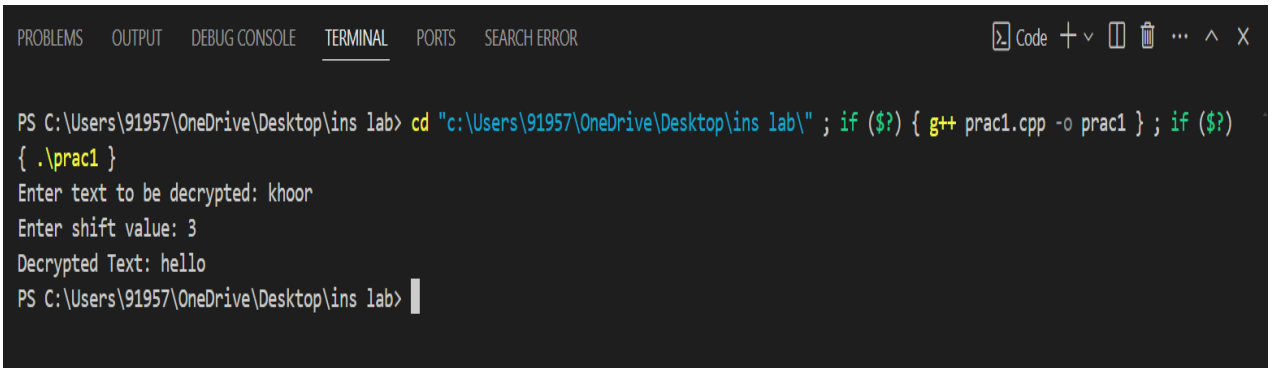
```
    getline(cin, text);

    cout << "Enter shift value: ";
    cin >> shift;

    string decryptedText = decryption(text, shift);
    cout << "Decrypted Text: " << decryptedText << endl;

    return 0;
}
```

Output:



HARDIK GODKAR
210303105029
7B23

Practical – 2

Aim:- Monoalphabetic Cipher

Theory:

• A monoalphabetic cipher is a substitution cipher where each letter in the plaintext is replaced by a corresponding letter from a fixed substitution alphabet.
• This method ensures that each letter consistently maps to the same substitute letter throughout the message.
• Although simple to implement, monoalphabetic ciphers are highly vulnerable to frequency analysis, as the patterns in letter frequency remain unchanged from the plaintext to the ciphertext.
• Historical examples, like the Caesar cipher, demonstrate the fundamental concept of monoalphabetic encryption and its straightforward yet easily breakable nature.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
 char mp[26][2];
 int j = 25;

 for (int i = 0; i < 26; i++) {
 mp[i][0] = 'a' + i;
 mp[i][1] = 'a' + j;
 j--;
 }
```

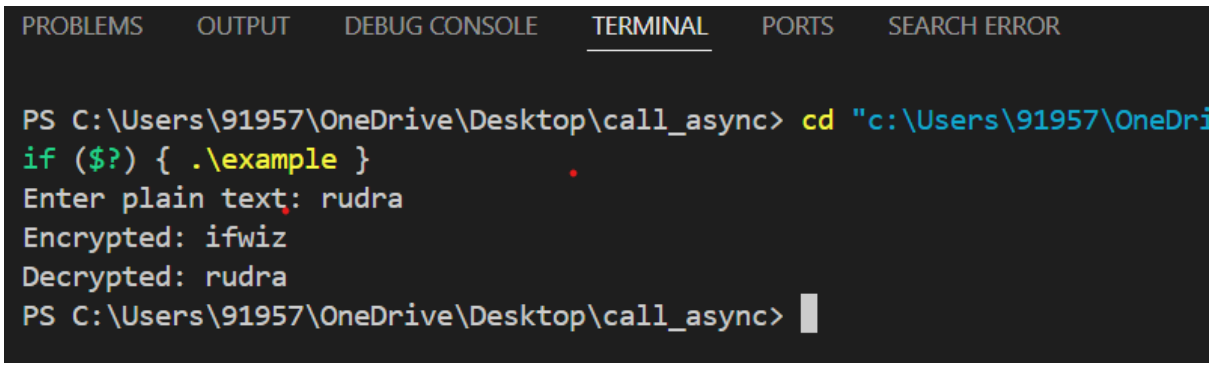

HARDIK GODKAR
210303105029
7B23

```
 printf("Enter plain text: ");
 char p[100];
 scanf("%s", p);
 char ans[100];
 int len = strlen(p);
for (int i = 0; i < len; i++) {
 for (int k = 0; k < 26; k++) {
 if (p[i] == mp[k][0]) {
 ans[i] = mp[k][1];
 break;
 }
 }
 }
 printf("Encrypted: %s\n", ans);
 char ans2[100];
 int len2 = strlen(ans);

 for (int i = 0; i < len2; i++) {
 for (int k = 0; k < 26; k++) {
 if (ans[i] == mp[k][1]) {
 ans2[i] = mp[k][0];
 break;
 }
 }
 }
 printf("Decrypted: %s\n", ans2);
 return 0;
}
```


HARDIK GODKAR
210303105029
7B23

Output:



HARDIK GODKAR
210303105029
7B23

Practical-3

Aim: A) Playfair Cipher Encryption

Theory:

The Playfair cipher was the first practical digraph substitution cipher.

- In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.
- Pair cannot be made with same letter.
- Break the letter in single and add a bogus letter to the previous letter.
- If the letter is standing alone in the process of pairing, then add an extra bogus letter with the alone letter.
- If both the letters are in the same column: Take the letter below each one (going back to the top if at the bottom).

Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 100

void toLowerCase(char *text) {
   for (int i = 0; text[i]; i++) {
      text[i] = tolower(text[i]);
   }
}

int removeSpaces(char *text) {
   int count = 0;
   for (int i = 0; text[i]; i++) {
      if (text[i] != ' ') {
```


HARDIK GODKAR
210303105029
7B23

```
            text[count++] = text[i];
        }
    }
    text[count] = '\0';
    return count;
}

void generateKeyTable(const char *key, char keyTable[5][5]) {
    int dicty[26] = {0};
    int i = 0, j = 0;

    for (int k = 0; key[k]; k++) {
        if (key[k] != 'j' && !dicty[key[k] - 'a']) {
            dicty[key[k] - 'a'] = 1;
            keyTable[i][j++] = key[k];
            if (j == 5) { i++; j = 0; }
        }
    }

    for (int k = 0; k < 26; k++) {
        if (!dicty[k] && k != 'j' - 'a') {
            keyTable[i][j++] = 'a' + k;
            if (j == 5) { i++; j = 0; }
        }
    }
}

void searchKeyTable(const char keyTable[5][5], char a, char b, int pos[]) {
    a = (a == 'j') ? 'i' : a;
    b = (b == 'j') ? 'i' : b;

    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
```


HARDIK GODKAR
210303105029
7B23

```
            if (keyTable[i][j] == a) { pos[0] = i; pos[1] = j; }
            if (keyTable[i][j] == b) { pos[2] = i; pos[3] = j; }
        }
    }
}

void encryptText(char *text, const char keyTable[5][5]) {
    int pos[4];

    for (int i = 0; text[i]; i += 2) {
        searchKeyTable(keyTable, text[i], text[i + 1], pos);

        if (pos[0] == pos[2]) {
            text[i] = keyTable[pos[0]][(pos[1] + 1) % 5];
            text[i + 1] = keyTable[pos[0]][(pos[3] + 1) % 5];
        } else if (pos[1] == pos[3]) {
            text[i] = keyTable[(pos[0] + 1) % 5][pos[1]];
            text[i + 1] = keyTable[(pos[2] + 1) % 5][pos[1]];
        } else {
            text[i] = keyTable[pos[0]][pos[3]];
            text[i + 1] = keyTable[pos[2]][pos[1]];
        }
    }
}

void prepareText(char *text) {
    int len = strlen(text);
    if (len % 2 != 0) {
        text[len] = 'z';
        text[len + 1] = '\0';
    }
}
```


HARDIK GODKAR
210303105029
7B23

```
void encryptByPlayfairCipher(char *text, const char *key) {
   char keyTable[5][5];
   toLowerCase(text);
   int ps = removeSpaces(text);
   toLowerCase((char *)key);  // Cast away const for compatibility
   int ks = removeSpaces((char *)key);
   prepareText(text);
   generateKeyTable(key, keyTable);
   encryptText(text, keyTable);
}

int main() {
   char str[SIZE] = "vishal";
   char key[SIZE] = "hill";
   printf("Key text: %s\n", key);
   printf("Plain text: %s\n", str);
   encryptByPlayfairCipher(str, key);
   printf("Cipher text: %s\n", str);
   return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Pr
ac/" && gcc Prac_3_
1.c -o Prac_3_1 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"Prac_3_1
Key text: hill
Plain text: vishal
Cipher text: whqlba
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

Aim: B) Playfair Cipher Decryption

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 30

void toLowerCase(char plain[], int ps) {
for(int i = 0; i < ps; i++) {

    if (plain[i] >= 'A' && plain[i] <= 'Z')

        plain[i] += 32;

    }
    }
int removeSpaces(char* plain, int ps) {
    int count = 0;
    for(int i = 0; i < ps; i++) {
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    }
        plain[count] = '\0';
        return count;
    }



    void generateKeyTable(char key[], int ks, char keyT[5][5]) {
    int dicty[26] = {0}, i = 0, j = 0;
    for(int k = 0; k < ks; k++) {
        if (key[k] != 'j' && !dicty[key[k] - 'a']) {
```


HARDIK GODKAR
210303105029
7B23

```
            dicty[key[k] - 'a'] = 1;
            keyT[i][j++] = key[k];
            if (j == 5) { i++; j = 0; }
        }
    }
    for (char ch = 'a'; ch <= 'z'; ch++) {
        if (ch != 'j' && !dicty[ch - 'a']) {
            keyT[i][j++] = ch;
            if (j == 5) { i++; j = 0; }
        }
    }
}

void search(char keyT[5][5], char a, char b, int arr[]) {
    if (a == 'j') a = 'i';
    if (b == 'j') b = 'i';
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (keyT[i][j] == a) { arr[0] = i; arr[1] = j; }
            if (keyT[i][j] == b) { arr[2] = i; arr[3] = j; }
        }
    }
}

int mod5(int a) {
    return (a + 5) % 5;
}

void decrypt(char str[], char keyT[5][5], int ps) {
    int a[4];
    for (int i = 0; i < ps; i += 2) {
        search(keyT, str[i], str[i + 1], a);
        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] - 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];
        } else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] - 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];
        } else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
```


HARDIK GODKAR
210303105029
7B23

```
        }
    }
}

void decryptByPlayfairCipher(char str[], char key[]) {
    int ks = removeSpaces(key, strlen(key));
    toLowerCase(key, ks);
    int ps = removeSpaces(str, strlen(str));
    toLowerCase(str, ps);
    char keyT[5][5];
    generateKeyTable(key, ks, keyT);
    decrypt(str, keyT, ps);
}

int main() {
    char str[SIZE] = "whqlba", key[SIZE] = "hill";
    printf("Key text: %s\n", key);
    printf("Cipher text: %s\n", str);
    decryptByPlayfairCipher(str, key);
    printf("Deciphered text: %s\n", str);
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/" && gcc Prac_3_2
.c -o Prac_3_2 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"Prac_3_2
Key text: hill
Cipher text: whqlba
Deciphered text: vishal
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

Practical-4

Aim: A) **Hill Cipher Encryption**

Theory:

- Hill cipher is a polygraphic substitution cipher based on linear algebra.
- Each letter is represented by a number modulo 26.
- To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26.
- To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.
- The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

Code:

```
#include <stdio.h>
#include <string.h>
```


HARDIK GODKAR
210303105029
7B23

```
void getKeyMatrix(char key[], int keyMatrix[3][3]) {
   int k = 0;
   for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
         keyMatrix[i][j] = key[k++] % 65;
}

void encrypt(int cipherMatrix[3][1], int keyMatrix[3][3], int messageVector[3][1]) {
   for (int i = 0; i < 3; i++) {
      cipherMatrix[i][0] = 0;
      for (int j = 0; j < 3; j++)
         cipherMatrix[i][0] += keyMatrix[i][j] * messageVector[j][0];
      cipherMatrix[i][0] %= 26;
   }
}

void HillCipher(char message[], char key[]) {
   int keyMatrix[3][3];
   getKeyMatrix(key, keyMatrix);

   int messageVector[3][1];
   for (int i = 0; i < 3; i++)
      messageVector[i][0] = message[i] % 65;

   int cipherMatrix[3][1];
   encrypt(cipherMatrix, keyMatrix, messageVector);

   char CipherText[4];
   for (int i = 0; i < 3; i++)
      CipherText[i] = cipherMatrix[i][0] + 65;
   CipherText[3] = '\0';

   printf("Ciphertext: %s\n", CipherText);
}

int main() {
   char message[] = "ACT";
   char key[] = "GYBNQKURP";
   HillCipher(message, key);
```


HARDIK GODKAR
210303105029
7B23

```
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS
Prac/" && gcc Prac_4_1.c -o Prac_4_1 && "/Users/vishalmaurya/Documents/Study/Sem 7/I
NS/INS  Prac/"Prac_4_1
Ciphertext: POH
vishalmaurya@192 INS  Prac %
```

Aim: B) Hill Cipher Decryption

Code:

```
#include <stdio.h>
#include <string.h>

void getKeyMatrix(char key[], int keyMatrix[3][3]) {
    for (int i = 0, k = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            keyMatrix[i][j] = key[k++] % 65;
}

void multiplyMatrices(int result[3][1], int matrix1[3][3], int matrix2[3][1]) {
    for (int i = 0; i < 3; i++) {
        result[i][0] = 0;
        for (int j = 0; j < 3; j++)
            result[i][0] += matrix1[i][j] * matrix2[j][0];
        result[i][0] %= 26;
```


HARDIK GODKAR
210303105029
7B23

```
    }
}

int modInverse(int a, int m) {
    for (int x = 1; x < m; x++)
        if ((a * x) % m == 1)
            return x;
    return 1;
}

int determinant(int matrix[3][3], int n) {
    int det = 0;
    if (n == 1)
        return matrix[0][0];
    int temp[3][3], sign = 1;
    for (int f = 0; f < n; f++) {
        for (int i = 1; i < n; i++) {
            for (int j = 0, col = 0; j < n; j++) {
                if (j != f)
                    temp[i-1][col++] = matrix[i][j];
            }
        }
        det += sign * matrix[0][f] * determinant(temp, n - 1);
        sign = -sign;
    }
    return det;
}

void adjoint(int matrix[3][3], int adj[3][3]) {
    int sign = 1, temp[3][3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int p = 0, row = 0; p < 3; p++) {
                if (p == i) continue;
                for (int q = 0, col = 0; q < 3; q++) {
                    if (q == j) continue;
                    temp[row][col++] = matrix[p][q];
                }
                row++;
```


HARDIK GODKAR
210303105029
7B23

```
            }
            adj[j][i] = (sign * determinant(temp, 2)) % 26;
            if (adj[j][i] < 0) adj[j][i] += 26;
            sign = -sign;
        }
    }
}

void inverse(int matrix[3][3], int inverse[3][3]) {
    int det = determinant(matrix, 3);
    int invDet = modInverse(det % 26, 26);
    int adj[3][3];
    adjoint(matrix, adj);
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            inverse[i][j] = (adj[i][j] * invDet) % 26;
}

void HillCipherDecrypt(char cipher[], char key[]) {
    int keyMatrix[3][3], inverseKeyMatrix[3][3], cipherVector[3][1], messageVector[3][1];
    getKeyMatrix(key, keyMatrix);
    inverse(keyMatrix, inverseKeyMatrix);
    for (int i = 0; i < 3; i++)
        cipherVector[i][0] = cipher[i] % 65;
    multiplyMatrices(messageVector, inverseKeyMatrix, cipherVector);
    for (int i = 0; i < 3; i++)
        cipher[i] = messageVector[i][0] + 65;
}

int main() {
    char cipher[] = "POH";
    char key[] = "GYBNQKURP";
    HillCipherDecrypt(cipher, key);
    printf("Decrypted message: %s\n", cipher);
    return 0;
}
```

Output:

HARDIK GODKAR
210303105029
7B23

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS
Prac/" && gcc Prac_4_2.c -o Prac_4_2 && "/Users/vishalmaurya/Documents/Study/Sem 7/I
NS/INS  Prac/"Prac_4_2
Decrypted message: ACT
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

- **Encryption:** If the plaintext is "HELLO" and the key is "KEY," the key repeats to "KEYKE," and each letter is shifted accordingly: H (7) + K (10) = R (17), E (4) + E (4) = I (8), L (11) + Y (24) = J (9), and so on, forming the ciphertext "RIJVS."
- **Decryption:** The same key is used to reverse the shifts, restoring the original message.
- **Security:** This method makes frequency analysis attacks difficult, enhancing security compared to simple substitution ciphers.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

void vigenere_encrypt(char *plaintext, char *key) {
   int i, j;
   int keyLength = strlen(key);
   int textLength = strlen(plaintext);

   for (i = 0; i < keyLength; ++i) {
      key[i] = toupper(key[i]);
   }
   for (i = 0, j = 0; i < textLength; ++i, ++j) {
      if (j == keyLength) {
         j = 0;
      }
```


HARDIK GODKAR
210303105029
7B23

```
        if (isalpha(plaintext[i])) {
            char offset = isupper(plaintext[i]) ? 'A' : 'a';
            plaintext[i] = ((plaintext[i] - offset + key[j] - 'A') % 26) + offset;
        }
    }
}


int main() {
    char plaintext[1000];
    char key[100];

    printf("Enter plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = 0;
    printf("Enter key: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = 0;

    vigenere_encrypt(plaintext, key);
    printf("Encrypted text: %s\n", plaintext);
    return 0;
}
```


HARDIK GODKAR
210303105029
7B23

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/
" && gcc Prac_5_1.c -o Prac_5_1 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Pr
ac/"Prac_5_1
Enter plaintext: vishal maurya
Enter key: test
Encrypted text: omkatp ftyjrt
vishalmaurya@192 INS  Prac %
```

Aim: B) Polyalphabetic Cipher Decryption

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

void vigenere_decrypt(char *ciphertext, char *key) {
   int i, j;
   int keyLength = strlen(key);
   int textLength = strlen(ciphertext);

   for (i = 0; i < keyLength; ++i) {
      key[i] = toupper(key[i]);
   }

   for (i = 0, j = 0; i < textLength; ++i, ++j) {
      if (j == keyLength) {
         j = 0;
      }
      if (isalpha(ciphertext[i])) {
         char offset = isupper(ciphertext[i]) ? 'A' : 'a';
         ciphertext[i] = (((ciphertext[i] - offset) - (key[j] - 'A') + 26) % 26) + offset;
```


HARDIK GODKAR
210303105029
7B23

```
      }
    }
}

int main() {
    char ciphertext[1000];
    char key[100];

    printf("Enter ciphertext: ");
    fgets(ciphertext, sizeof(ciphertext), stdin);
    ciphertext[strcspn(ciphertext, "\n")] = 0;

    printf("Enter key: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = 0;

    vigenere_decrypt(ciphertext, key);
    printf("Decrypted text: %s\n", ciphertext);

    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/
" && gcc Prac_5_2.c -o Prac_5_2 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Pr
ac/"Prac_5_2
Enter ciphertext: omkatp ftyjrt
Enter key: test
Decrypted text: vishal maurya
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

Practical-6

Aim: A) Transposition Encryption

Theory:

The Rail Fence Cipher is a transposition cipher where the plaintext is written in a zigzag pattern across multiple "rails" (rows). To encrypt, the message is written diagonally across the chosen number of rails, then read row by row to form the ciphertext. Decryption reverses this by reconstructing the zigzag pattern and reading it to retrieve the original message. This method is simple and relies on rearranging the characters rather than altering them.

Code:

```
#include <stdio.h>
#include <string.h>

void rail_fence_encrypt(const char *plaintext, int num_rails, char *ciphertext) {
   if (num_rails <= 1) {
      strcpy(ciphertext, plaintext);
      return;
   }
   int len = strlen(plaintext);
   char rails[num_rails][len];

   for (int i = 0; i < num_rails; i++) {
      for (int j = 0; j < len; j++) {
         rails[i][j] = '\0';
      }
   }
   int rail_direction = 1;
   int current_rail = 0;

   for (int i = 0; i < len; i++) {
      rails[current_rail][i] = plaintext[i];
```


HARDIK GODKAR
210303105029
7B23

```
        current_rail += rail_direction;
        if (current_rail == 0 || current_rail == num_rails - 1) {
            rail_direction *= -1;
        }
    }
    int index = 0;
    for (int i = 0; i < num_rails; i++) {
        for (int j = 0; j < len; j++) {
            if (rails[i][j] != '\0') {
                ciphertext[index++] = rails[i][j];
            }
        }
    }
    ciphertext[index] = '\0';
}
int main() {
    const char plaintext[] = "vishal";
    int num_rails = 3;
    char ciphertext[strlen(plaintext) + 1];

    printf("Plaintext: %s\n", plaintext);
    rail_fence_encrypt(plaintext, num_rails, ciphertext);
    printf("Ciphertext: %s\n", ciphertext);

    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  P
rac/" && gcc Pra
c_6_1.c -o Prac_6_1 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"Prac
_6_1
Plaintext: vishal
Ciphertext: vaihls
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

Aim: B) Transposition Decryption

Code:

```
#include <stdio.h>
#include <string.h>

void rail_fence_decrypt(const char *ciphertext, int num_rails, char *plaintext) {
   if (num_rails <= 1) {
      strcpy(plaintext, ciphertext);
      return;
   }

   int len = strlen(ciphertext);
   int rail_length[num_rails];
   memset(rail_length, 0, sizeof(rail_length));
   int rail_direction = 1;
   int current_rail = 0;

   for (int i = 0; i < len; i++) {
      rail_length[current_rail]++;
      current_rail += rail_direction;
      if (current_rail == 0 || current_rail == num_rails - 1) {
         rail_direction *= -1;
```


HARDIK GODKAR
210303105029
7B23

```
    }
}

char rails[num_rails][len];
int index = 0;
for (int i = 0; i < num_rails; i++) {
    for (int j = 0; j < rail_length[i]; j++) {
        rails[i][j] = ciphertext[index++];
    }
}
rail_direction = 1;
current_rail = 0;
int rail_index[num_rails];
memset(rail_index, 0, sizeof(rail_index));

int plaintext_index = 0;
for (int i = 0; i < len; i++) {
    plaintext[plaintext_index++] = rails[current_rail][rail_index[current_rail]++];
    current_rail += rail_direction;
    if (current_rail == 0 || current_rail == num_rails - 1) {
        rail_direction *= -1;
    }
}
plaintext[plaintext_index] = '\0';
```


HARDIK GODKAR
210303105029
7B23

```
}
int main() {
   const char ciphertext[] = "vaihls";
   int num_rails = 3;
   char plaintext[strlen(ciphertext) + 1];
   printf("Ciphertext: %s\n", ciphertext);
   rail_fence_decrypt(ciphertext, num_rails, plaintext);
   printf("Plaintext: %s\n", plaintext);
   return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS
 Prac/" && gcc Prac_6_2.c -o Prac_6_2 && "/Users/vishalmaurya/Documents/Study/Sem 7
/INS/INS  Prac/"Prac_6_2
Ciphertext: vaihls
Plaintext: vishal
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

Practical-7

AIM: Implement Diffie- Hellman Key exchange Method.

Theory:

Steps for Diffie-Hellman Key Exchange

1. Agree on Public Parameters:

- o A prime number p
- o A base (or generator) g

2. Generate Private Keys:

- o Each party generates a private key.

3. Compute Public Keys:

- o Each party computes their public key using the formula:

public_key=g^private_key mod p

4. Exchange Public Keys:

- o Each party sends their public key to the other party.

5. Compute the Shared Secret:

- o Each party computes the shared secret using the received public key and their

own private key:

- o shared_secret= received_public_key^private_key mod p

Code:

```
#include <stdio.h>

int power(int a, int b, int p) {

   int result = 1;
```


HARDIK GODKAR
210303105029
7B23

```
    a = a % p;
    while (b > 0) {
        if (b % 2 == 1) {
            result = (result * a) % p;
        }
        b = b >> 1;
        a = (a * a) % p;
    }
    return result;
}

int main() {
    int P = 23;
    printf("The value of P: %d\n", P);
    int G = 9;
    printf("The value of G: %d\n", G);
    int a = 4;
    printf("The private key a for Alice: %d\n", a);
    int x = power(G, a, P);
    int b = 3;
    printf("The private key b for Bob: %d\n", b);
    int y = power(G, b, P);
    int ka = power(y, a, P);
    int kb = power(x, b, P);
```


HARDIK GODKAR
210303105029
7B23

```
    printf("Secret key for Alice is: %d\n", ka);
    printf("Secret key for Bob is: %d\n", kb);
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS
 Prac/" && gcc Prac_7.c -o Prac_7 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS
/INS  Prac/"Prac_7
The value of P: 23
The value of G: 9
The private key a for Alice: 4
The private key b for Bob: 3
Secret key for Alice is: 9
Secret key for Bob is: 9
```


HARDIK GODKAR
210303105029
7B23

PRACTICAL-8

AIM:(A) Implement One time pad encryption.

Theory:

1. Basic Principle: The key must be random, as long as the message, and used only once to produce ciphertext by combining it with the plaintext.

2. Encryption Process:: XOR is commonly used for binary data, and modular arithmetic for alphabetic characters to encrypt and decrypt.

3. Properties: OTP provides perfect secrecy with a truly random key used only once, ensuring symmetric encryption and key non-reusability.

4. Key Generation: The key must be as long as the plaintext, truly random, and ideally generated by hardware-based random number generators.

5. Security Considerations: The key must be truly random, securely distributed, and properly stored to prevent unauthorized decryption.

6. Vulnerabilities: Reusing the key or poor key management can lead to vulnerabilities like crib dragging attacks and practical difficulties.

7. Practical Use Cases: OTP is rarely used in modern cryptography but may be employed for highly secure, small-scale applications

HARDIK GODKAR
210303105029
7B23

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void generate_key(int length, char *key) {
    srand(time(0));
    for (int i = 0; i < length; i++) {
        key[i] = (rand() % 2) ? '1' : '0';
    }
    key[length] = '\0';
}

void string_to_binary(const char *text, char *binary) {
    while (*text) {
        unsigned char c = *text++;
        for (int i = 7; i >= 0; i--) {
            binary[i] = (c & 1) ? '1' : '0';
            c >>= 1;
        }
        binary += 8;
    }
    *binary = '\0';
```


HARDIK GODKAR
210303105029
7B23

```
}

void binary_to_string(const char *binary, char *text) {
    while (*binary) {
        unsigned char c = 0;
        for (int i = 0; i < 8; i++) {
            c = (c << 1) | (*binary++ - '0');
        }
        *text++ = c;
    }
    *text = '\0';
}

void xor_binary(const char *binary1, const char *binary2, char *result) {
    while (*binary1) {
        *result++ = (*binary1++ != *binary2++) ? '1' : '0';
    }
    *result = '\0';
}

void one_time_pad_encrypt(const char *plaintext, const char *key, char *ciphertext) {
    char binary_plaintext[1024];
    string_to_binary(plaintext, binary_plaintext);
    xor_binary(binary_plaintext, key, ciphertext);
```


HARDIK GODKAR
210303105029
7B23

```
}

void one_time_pad_decrypt(const char *ciphertext, const char *key, char *decrypted_text) {
    char decrypted_binary[1024];
    xor_binary(ciphertext, key, decrypted_binary);
    binary_to_string(decrypted_binary, decrypted_text);
}

int main() {
    const char *plaintext = "i am vishal";
    char binary_plaintext[1024];
    char key[1024];
    char ciphertext[1024];
    char decrypted_text[1024];

    string_to_binary(plaintext, binary_plaintext);
    generate_key(strlen(binary_plaintext), key);
    one_time_pad_encrypt(plaintext, key, ciphertext);

    printf("Plaintext: %s\n", plaintext);
    printf("Generated Key: %s\n", key);
    printf("Ciphertext (binary): %s\n", ciphertext);

    one_time_pad_decrypt(ciphertext, key, decrypted_text);
```


HARDIK GODKAR
210303105029
7B23

```
    printf("Decrypted Text: %s\n", decrypted_text);

    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/" &&
 gcc Prac_8_1.c -o Prac_8_1 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"Prac
_8_1
Plaintext: i am vishal
Generated Key: 1100011011101000010100101101001100010100110100011111101001010000001001001111
1101110111
Ciphertext (binary): 1010111111001000001100111011111000110100101001111001001100100011011110100
0001101000011011
Decrypted Text: i am vishal
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

AIM:(B) Implement One Time Pad Decryption.

Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void binary_to_text(const char *binary_str, char *result) {
    size_t length = strlen(binary_str);
    for (size_t i = 0; i < length; i += 8) {
        char byte[9];
        strncpy(byte, binary_str + i, 8);
        byte[8] = '\0';
        unsigned char character = (unsigned char) strtol(byte, NULL, 2);
        *result++ = character;
    }
    *result = '\0';
}
void otp_decrypt(const char *ciphertext, const char *key, char *plaintext) {
    if (strlen(ciphertext) != strlen(key)) {
        fprintf(stderr, "Ciphertext and key must be of the same length.\n");
        exit(EXIT_FAILURE);
    }
    char plaintext_binary[1024];
    size_t length = strlen(ciphertext);
    for (size_t i = 0; i < length; ++i) {
```


HARDIK GODKAR
210303105029
7B23

```
        plaintext_binary[i] = (ciphertext[i] != key[i]) ? '1' : '0';
    }
    plaintext_binary[length] = '\0';
    binary_to_text(plaintext_binary, plaintext);
}
int main() {
    const char *ciphertext =
"10101111110010000011001110111110001101001010011110010011001000110111101000011010000011011";
  const char *key =
"11000110111010000101001011010011000101001101000111111010010100000000100100111101101110111";
    char plaintext[1024];
    otp_decrypt(ciphertext, key, plaintext);
    printf("Decrypted plaintext: %s\n", plaintext);
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 && gcc Prac_8_2.c -o Prac_8_2 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 Prac_8_2
 Decrypted plaintext: i am vishal
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

PRACTICAL-9

AIM:(A) Implement RSA encryption algorithm.

Theory:

RSA is an asymmetric encryption algorithm that uses a pair of keys: a public key for encryption and a private key for decryption. The key generation process involves selecting two large prime numbers, computing their product to form the modulus, and choosing a public exponent along with a private exponent that ensures specific mathematical properties. Encryption is done by applying an exponentiation operation to the message using the public key, resulting in ciphertext. The security of RSA relies on the difficulty of factoring the product of the two primes. While encryption is generally efficient, decryption is slower but can be optimized. RSA is used in various applications, including digital signatures, secure communications in protocols like TLS/SSL, and Public Key Infrastructure (PKI) for managing digital certificates.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
int gcd(int a, int b) {
   while (b != 0) {
      int temp = b;
      b = a % b;
      a = temp;
   }
```


HARDIK GODKAR
210303105029
7B23

```
    return a;
}
int modinv(int e, int phi) {
    int d = 0, x1 = 0, x2 = 1, y1 = 1;
    int temp_phi = phi;
    while (e > 0) {
        int temp1 = temp_phi / e;
        int temp2 = temp_phi - temp1 * e;
        temp_phi = e;
        e = temp2;
        int x = x2 - temp1 * x1;
        int y = d - temp1 * y1;
        x2 = x1;
        x1 = x;
        d = y1;
        y1 = y;
    }
    if (temp_phi == 1) {
        return d + phi;
    }
    return d;
}
void generate_keypair(int p, int q, int *e, int *d, int *n) {
    *n = p * q;
```


HARDIK GODKAR
210303105029
7B23

```
    int phi = (p - 1) * (q - 1);
    srand(time(0));
    *e = rand() % (phi - 1) + 1;
    while (gcd(*e, phi) != 1) {
        *e = rand() % (phi - 1) + 1;
    }
    *d = modinv(*e, phi);
}
void encrypt(int key, int n, const char *plaintext, int *cipher, int *cipher_len) {
    int i = 0;
    while (plaintext[i] != '\0') {
        cipher[i] = (int)pow((int)plaintext[i], key) % n;
        i++;
    }
    *cipher_len = i;
}
int main() {
    int p = 61;
    int q = 53;
    int e, d, n;
    generate_keypair(p, q, &e, &d, &n);
    printf("Public key: (%d, %d)\n", e, n);
    printf("Private key: (%d, %d)\n", d, n);
    const char *message = "HELLO VISHAL";
```


HARDIK GODKAR
210303105029
7B23

```
    int encrypted_message[256];
    int cipher_len;
    encrypt(e, n, message, encrypted_message, &cipher_len);
    printf("Original message: %s\n", message);
    printf("Encrypted message: ");
    for (int i = 0; i < cipher_len; i++) {
        printf("%d ", encrypted_message[i]);
    }
    printf("\n");
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 && gcc Prac_9_1.c -o Prac_9_1 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 Prac_9_1
 Public key: (769, 3233)
 Private key: (4609, 3233)
 Original message: HELLO VISHAL
 Encrypted message: 2193 2193 2193 2193 2193 2193 2193 2193 2193 2193 2193 2193
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

AIM(B): Implement RSA decryption algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
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 >> 1;
      base = (base * base) % mod;
   }
   return result;
}
int gcd(int a, int b) {
   while (b != 0) {
      int temp = b;
      b = a % b;
      a = temp;
   }
   return a;
}
```


HARDIK GODKAR
210303105029
7B23

```
int modinv(int e, int phi) {
    int d = 0, x1 = 0, x2 = 1, y1 = 1;
    int temp_phi = phi;
    while (e > 0) {
        int temp1 = temp_phi / e;
        int temp2 = temp_phi - temp1 * e;
        temp_phi = e;
        e = temp2;
        int x = x2 - temp1 * x1;
        int y = d - temp1 * y1;
        x2 = x1;
        x1 = x;
        d = y1;
        y1 = y;
    }
    if (temp_phi == 1) {
        return d + phi;
    }
    return d;
}
void generate_keypair(int p, int q, int* e, int* d, int* n) {
    int phi = (p - 1) * (q - 1);
    *n = p * q;
```


HARDIK GODKAR
210303105029
7B23

```
    srand(time(0));
    *e = rand() % (phi - 1) + 1;
    while (gcd(*e, phi) != 1) {
        *e = rand() % (phi - 1) + 1;
    }
    *d = modinv(*e, phi);
}
void encrypt(int e, int n, const char* plaintext, int* ciphertext, int len) {
    for (int i = 0; i < len; i++) {
        ciphertext[i] = mod_exp(plaintext[i], e, n);
    }
}
void decrypt(int d, int n, int* ciphertext, char* plaintext, int len) {
    for (int i = 0; i < len; i++) {
        plaintext[i] = (char)mod_exp(ciphertext[i], d, n);
    }
    plaintext[len] = '\0';
}
int main() {
    int p = 61;
    int q = 53;
    int e, d, n;
    generate_keypair(p, q, &e, &d, &n);
    printf("Public key: (%d, %d)\n", e, n);
```


HARDIK GODKAR
210303105029
7B23

```
    printf("Private key: (%d, %d)\n", d, n);
    char message[] = "HELLO VISHAL";
    int len = sizeof(message) - 1;
    int encrypted_message[len];
    char decrypted_message[len + 1];
    encrypt(e, n, message, encrypted_message, len);
    printf("Encrypted message: ");
    for (int i = 0; i < len; i++) {
        printf("%d ", encrypted_message[i]);
    }
    printf("\n");
    decrypt(d, n, encrypted_message, decrypted_message, len);
    printf("Decrypted message: %s\n", decrypted_message);
    return 0;
}
```

Output:

```
vishalmaurya@192 INS  Prac % cd "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 && gcc Prac_9_2.c -o Prac_9_2 && "/Users/vishalmaurya/Documents/Study/Sem 7/INS/INS  Prac/"
 Prac_9_2
 Public key: (67, 3233)
 Private key: (1723, 3233)
 Encrypted message: 3039 521 83 83 1889 1676 561 1506 1825 3039 768 83
 Decrypted message: HELLO VISHAL
vishalmaurya@192 INS  Prac %
```


HARDIK GODKAR
210303105029
7B23

PRACTICAL-10

AIM: Demonstrate working of digital signature using cryptool.

Theory:

1. Overview of Digital Signatures

- **Purpose**: Digital signatures provide authenticity, integrity, and non-repudiation to digital communications and documents.
- **Components**: A digital signature process involves a key pair (public and private keys) and a hashing algorithm.

2. Key Concepts

- **Public and Private Keys**:
  - **Private Key**: Used to generate the signature.
  - **Public Key**: Used to verify the signature.
- **Hash Function**: Converts the original data into a fixed-size hash value. Common hash functions include SHA-256 and SHA-512.
- **Signature Generation**: The hash of the message is encrypted with the sender's private key to create a signature.
- **Signature Verification**: The recipient decrypts the signature with the sender's public key to retrieve the hash, then hashes the received message and compares it with the decrypted hash.

3. Steps to Demonstrate Digital Signatures Using CrypTool

1. **Generate Key Pair**:
   - **Generate a Public-Private Key Pair**:
     - Open CrypTool and select the digital signature or cryptographic module.
     - Generate a key pair using a cryptographic algorithm such as RSA or ECDSA.
     - Save the private key securely and the public key for verification purposes.
2. Create and Sign a Document:
   - **Prepare the Document**:
     - Write or load the document or message you want to sign.
   - Hash the Document:
     - CrypTool will automatically hash the document using a chosen hash algorithm (e.g., SHA-256).

HARDIK GODKAR
210303105029
7B23

3. Verify the Signature:
    - **Load the Document and Signature**:
        - Open CrypTool and load the document and its attached signature.
    - Hash the Document:
        - Compute the hash of the received document using the same hash function.
    - Decrypt the Signature:
        - Use the sender's public key to decrypt the signature and retrieve the original hash.
    - Compare Hashes:
        - Compare the computed hash with the decrypted hash. If they match, the signature is valid, confirming the document's integrity and authenticity.

4. Key Properties

- **Authenticity**: The digital signature verifies that the document was signed by the claimed sender.
- **Integrity**: Ensures that the document has not been altered after signing.
- **Non-repudiation**: The sender cannot deny the authenticity of the signature.

5. Use Cases

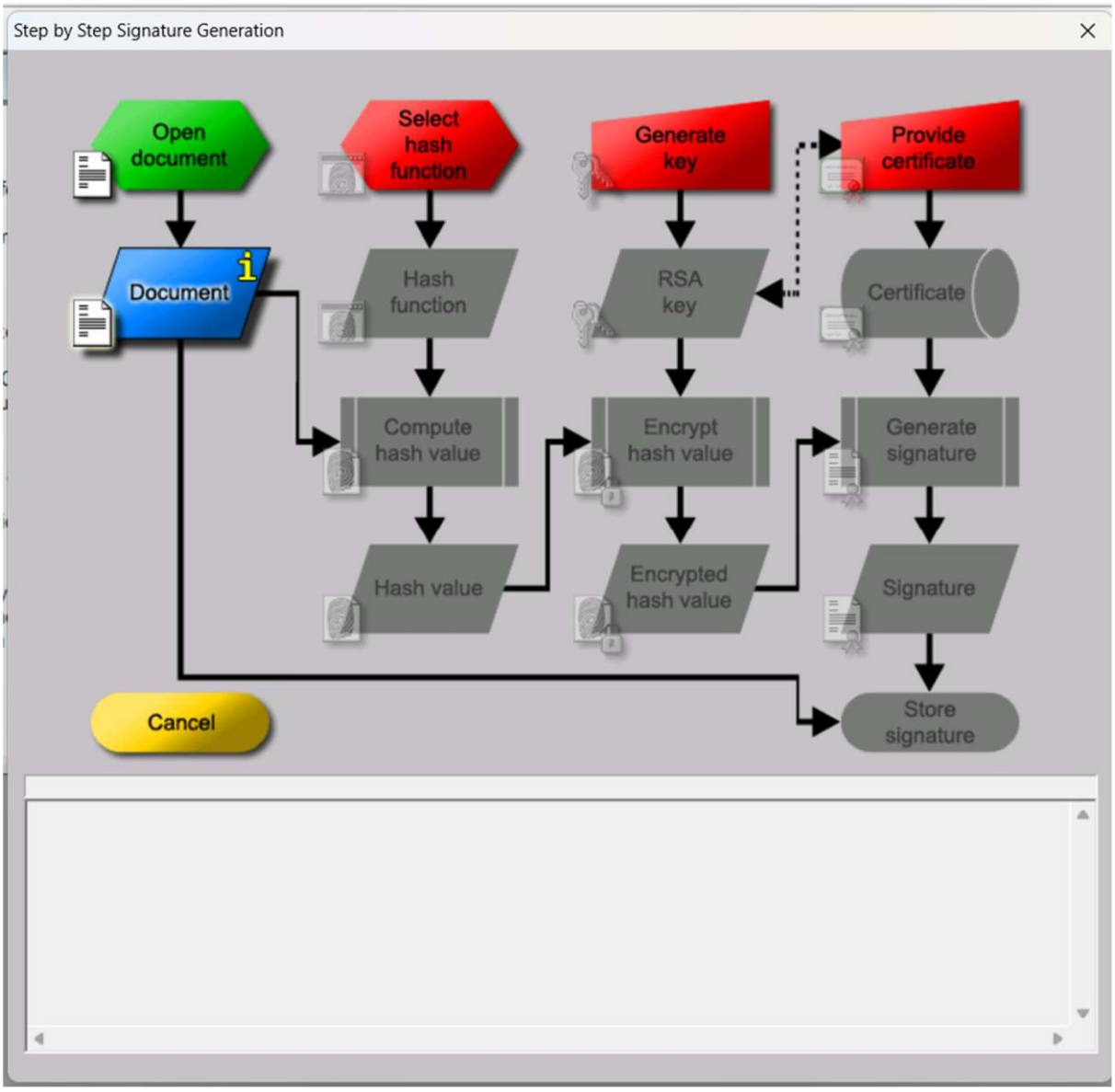
- **Email**: Digitally sign emails to confirm the sender's identity and ensure that the message hasn't been tampered with.
- **Software Distribution**: Sign software to verify the source and integrity of the software package.
- **Legal Documents**: Digitally sign contracts and legal documents for secure and verifiable transactions.

6. Security Considerations

- **Key Management**: Ensure private keys are kept secure and confidential.
- **Hash Function Choice**: Use strong, collision-resistant hash functions to prevent hash collisions.
- **Algorithm Choice**: Choose robust cryptographic algorithms with proven security properties.

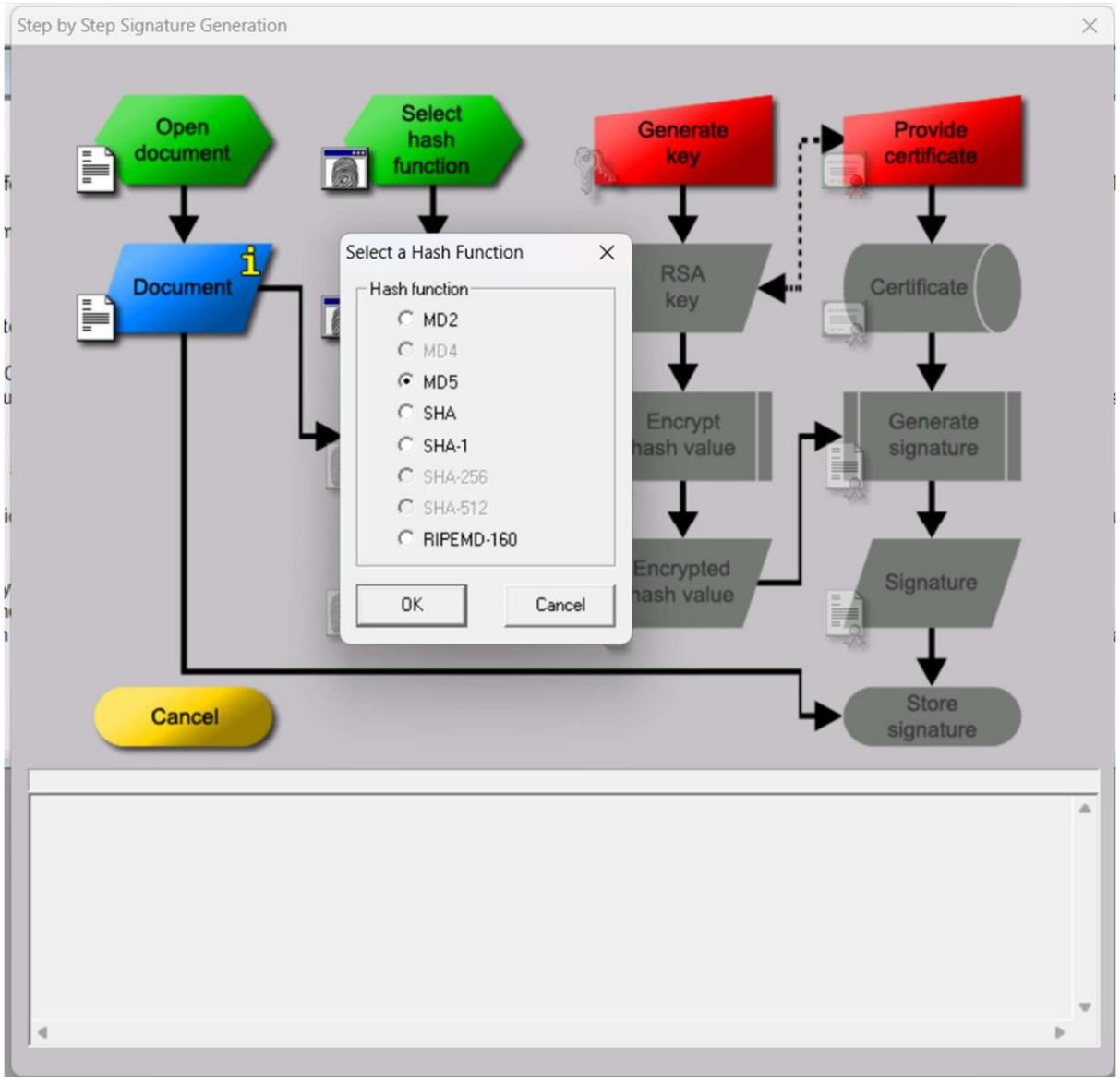
HARDIK GODKAR
210303105029
7B23

1.Open a document name file_name.txt.



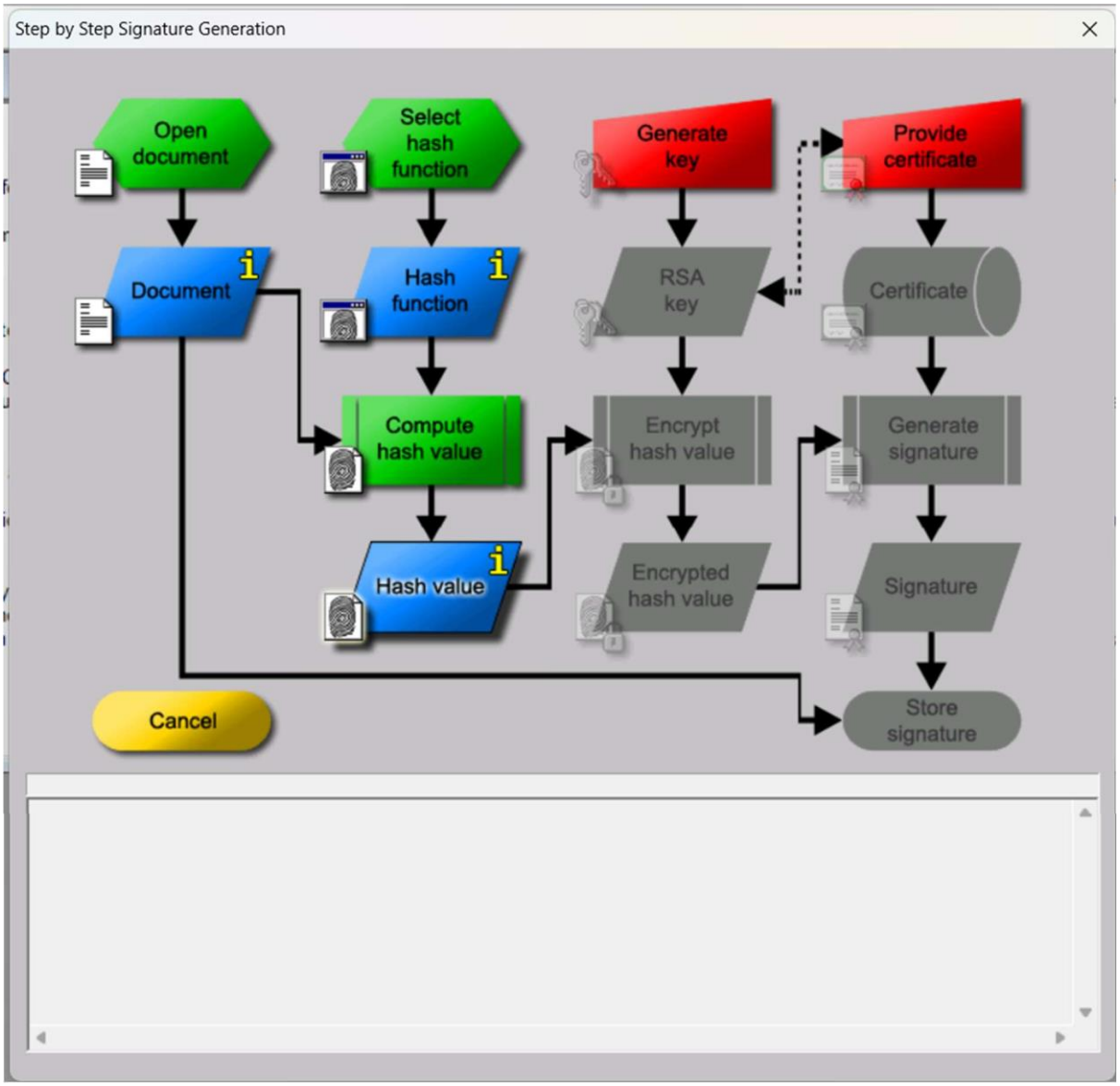
HARDIK GODKAR
210303105029
7B23

2.Select a Hash function.



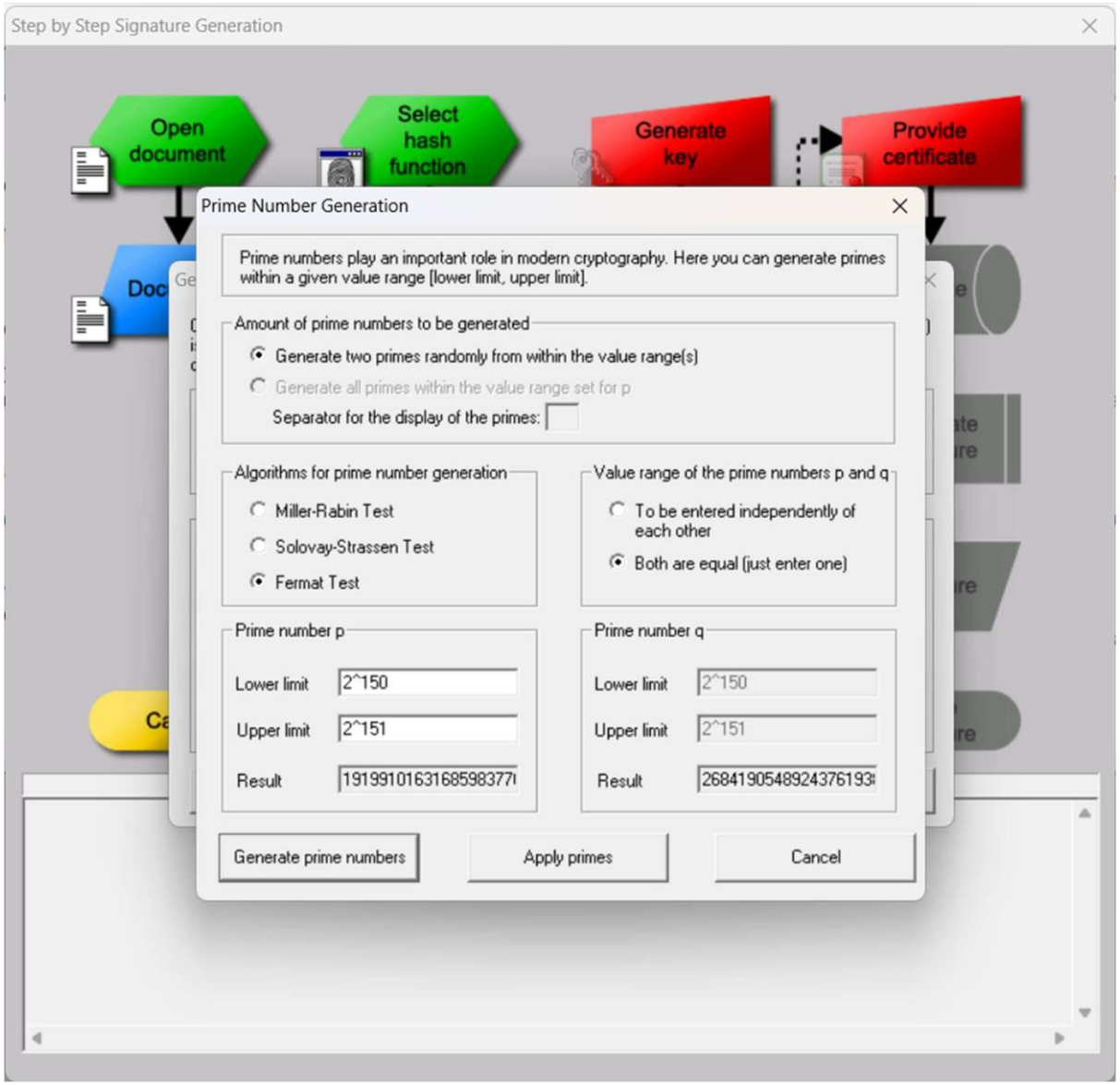
HARDIK GODKAR
210303105029
7B23

3.Compute Hash Value.



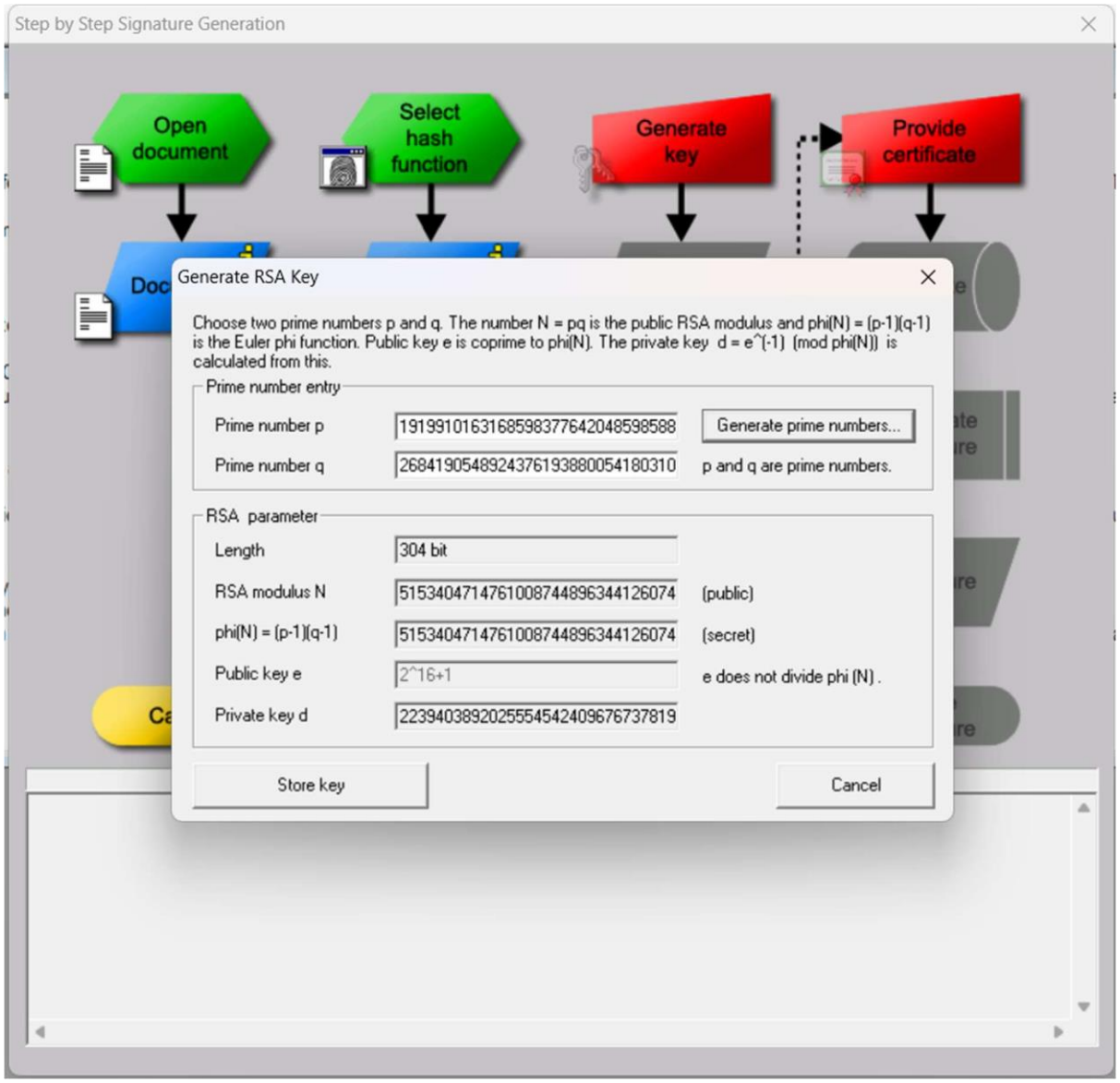
HARDIK GODKAR
210303105029
7B23

4. Generate key.



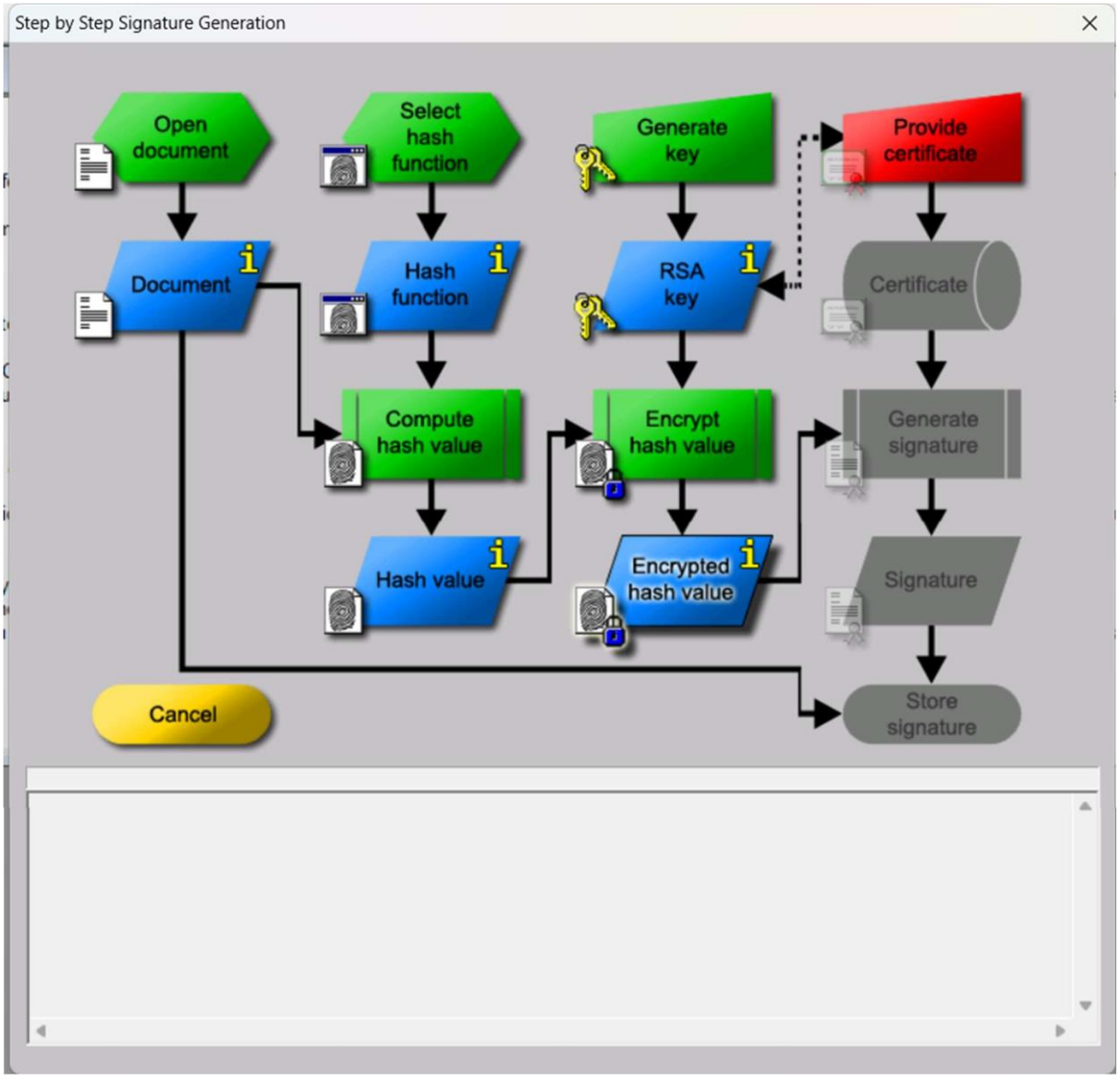
HARDIK GODKAR
210303105029
7B23

5. Store Keys.



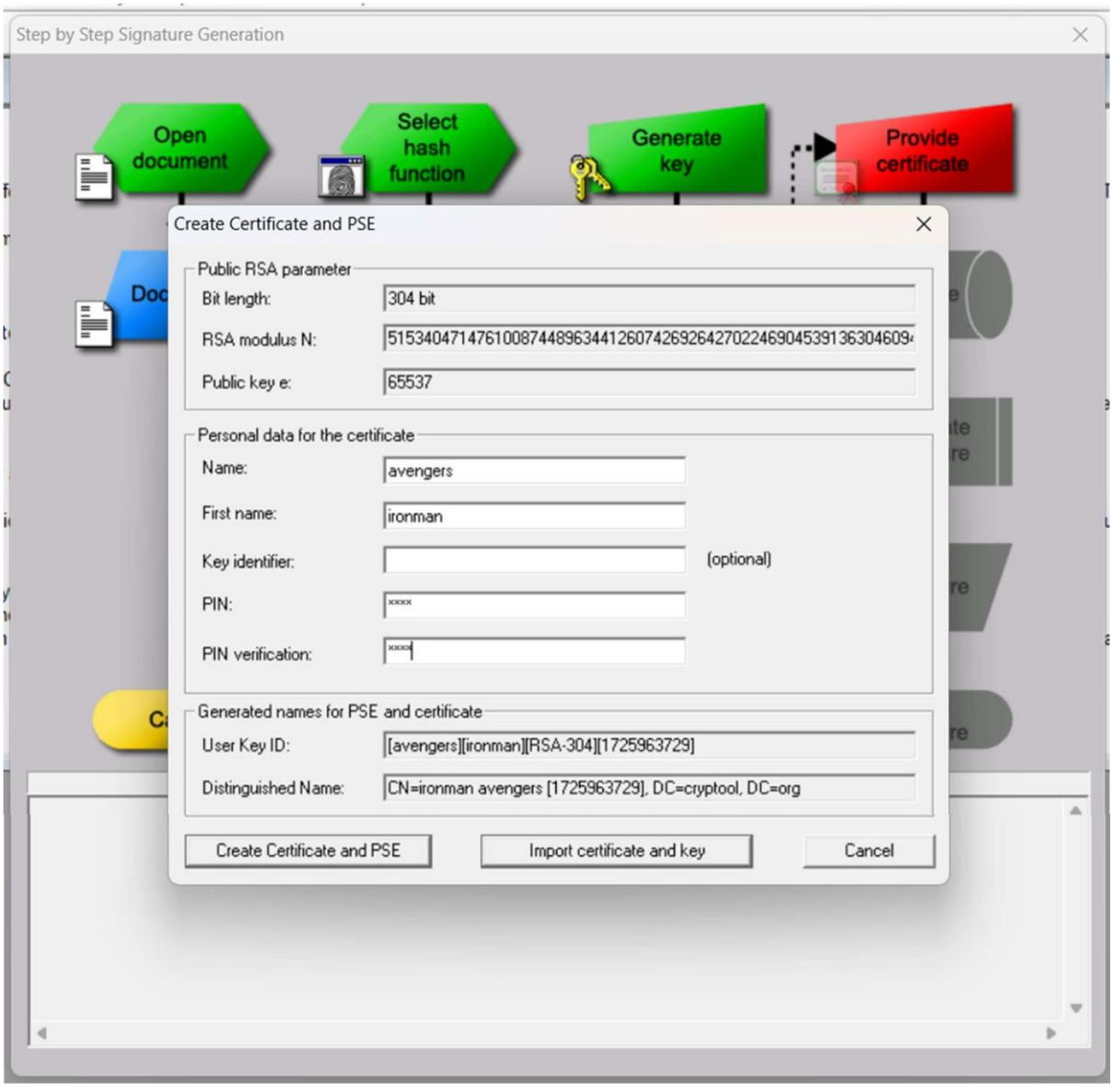
HARDIK GODKAR
210303105029
7B23

6.Encrypt Hash Value



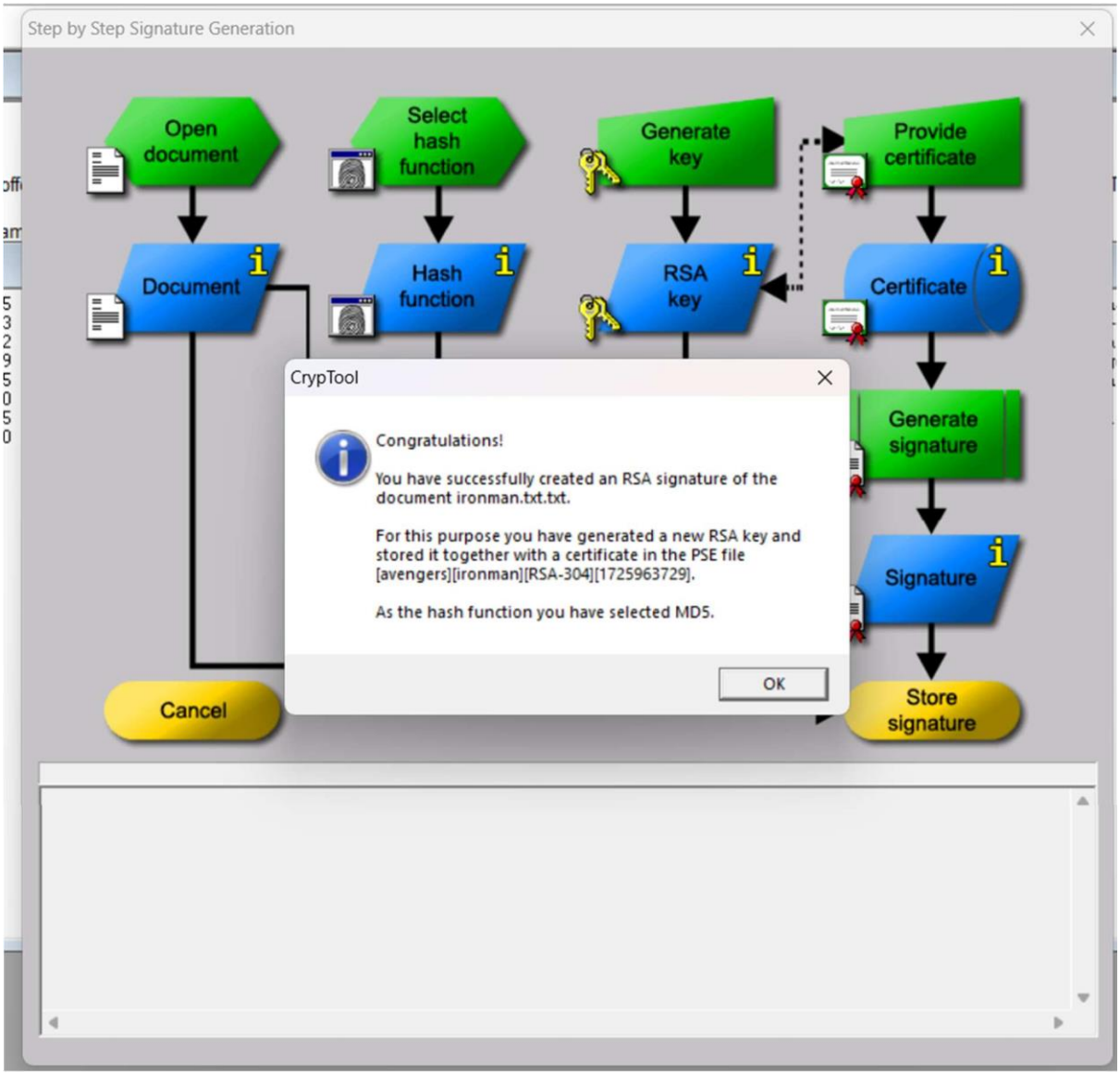
HARDIK GODKAR
210303105029
7B23

7.Provide certificate.



HARDIK GODKAR
210303105029
7B23

8.Store Signature.



Output:

HARDIK GODKAR
210303105029
7B23

HARDIK GODKAR
210303105029
7B23