ASYMMETRIC ENCRYPTION

REPORT

PREPARED BY:

BOUMEDIENE DHOHA
CHELIHI HADJER
LARBI HAYAT
SALMI AMANI

SUBMISSION:
5TH JANUARY 2025

THIS REPORT PRESENTS THE IMPLEMENTATION AND VISUALIZATION OF KEY ASYMMETRIC ENCRYPTION ALGORITHMS: RSA, HELLMAN, GOLDWASSER-MICALLI AND ELLIPTIC CURVE CRYPTOGRAPHY (ECC).
IT EXPLORES THE MATHEMATICAL FOUNDATIONS, IMPLEMENTATION CHALLENGES, AND GRAPHICAL REPRESENTATIONS OF THESE CRYPTOGRAPHIC TECHNIQUES. THE PROJECT DEMONSTRATES HOW THESE ALGORITHMS ENSURE SECURE COMMUNICATION THROUGH PUBLIC-KEY CRYPTOGRAPHY.
THE IMPLEMENTATION WAS WITH C LANGUAGE AND SDL LIBRARY IN VISUAL STUDIO CODE. IN ADDITION TO USING A LOT OF RESOURCES RELATED TO THE TOPIC MENTIONED ABOVE .

INTRODUCTION

Encryption is the process of converting information or data into a secure format so that unauthorized parties cannot access or read it. This is done using algorithms and keys. Encryption is widely used in cybersecurity to protect sensitive information, ensuring that data transmitted over the internet remains private and secure.

TYPES OF ENCRYPTION

- **SYMMETRIC ENCRYPTION (PRIVATE-KEY ENCRYPTION):**
  - In symmetric encryption, the same key is used for both encryption and decryption.
  - The key must be kept secret and shared between the sender and receiver.
  - Examples: Advanced Encryption Standard (AES), Data Encryption Standard (DES), and RC4.

- **ASYMMETRIC ENCRYPTION (PUBLIC-KEY ENCRYPTION):**
  - Uses a pair of keys: a public key and a private key.
  - The public key is used for encryption, and the private key is used for decryption.
  - The public key can be shared openly, while the private key remains secret.
  - Examples: RSA, Elliptic Curve Cryptography (ECC), and Diffie-Hellman key exchange.

ENCRYPTION PROCESS

- Plaintext: The original, readable data.
- Encryption Algorithm: The mathematical process that transforms plaintext into ciphertext.
- Ciphertext: The scrambled, unreadable form of the plaintext.
- Decryption Algorithm: The reverse of the encryption algorithm, which converts ciphertext back into plaintext.

USE CASES OF ENCRYPTION

- Secure Communication: Encrypting messages to prevent eavesdropping.
- Data Storage: Protecting sensitive data stored in databases or files.
- Digital Signatures: Verifying the authenticity and integrity of data.
- Cryptocurrency: Securing transactions on blockchain networks.

KEY CONCEPTS

- Key: A piece of information used in the encryption and decryption process. It could be a simple password or a complex mathematical value.
- Hashing: A one-way encryption process that transforms data into a fixed-size string (a hash), which is irreversible.

Encryption ensures that even if data is intercepted, it cannot be read or tampered with unless the key is known. This makes it a fundamental component of modern security protocols, including SSL/TLS for secure web communication and VPNs for private internet connections.

ASYMMETRIC ENCRYPTION

Asymmetric encryption, also known as public-key encryption, is a cryptographic technique that uses a pair of keys: a public key and a private key. These keys are mathematically linked, but it's computationally infeasible to derive the private key from the public key.
How It Works:

PUBLIC KEY:

This key is publicly shared with anyone. It’s used to encrypt data. The public key cannot be used to decrypt the data it encrypts.

PRIVATE KEY:

This key is kept secret and is used to decrypt data encrypted with the corresponding public key. Only the holder of the private key can decrypt the data.

ENCRYPTION AND DECRYPTION PROCESS:

1. ENCRYPTION:

- The sender encrypts the message (plaintext) using the recipient’s public key.
- Once encrypted, the message becomes ciphertext, which is unreadable without the private key.

2. DECRYPTION:

- The recipient uses their private key to decrypt the ciphertext back into plaintext.

Since the public key is distributed openly, anyone can send a secure message to the key owner, but only the key owner can decrypt the message using their private key.

APPLICATIONS OF ASYMMETRIC ENCRYPTION:

- Secure Communication: It’s used in protocols like SSL/TLS (for HTTPS) to secure communication between web browsers and servers.
- Digital Signatures: A message can be signed with the sender's private key, and anyone can verify the authenticity using the sender’s public key.
- Email Encryption: Services like PGP (Pretty Good Privacy) and S/MIME use asymmetric encryption for secure email communication.
- Cryptocurrencies: Public keys are used to receive cryptocurrency (e.g., Bitcoin), and private keys are used to sign transactions.

ADVANTAGES:

- Security: The private key never needs to be transmitted, so there’s no risk of it being intercepted.
- Non-repudiation: Digital signatures can be used to verify the authenticity and integrity of messages, ensuring the sender cannot deny having sent it.
- Scalability: Asymmetric encryption only requires the public key to be shared widely, making it more practical in large-scale systems.

DISADVANTAGES:

- Performance: Asymmetric encryption is computationally heavier than symmetric encryption, which can make it slower. This is why it’s often used in conjunction with symmetric encryption (e.g., in SSL/TLS protocols).

HYBRID ENCRYPTION:

Asymmetric encryption is often combined with symmetric encryption to improve efficiency. The asymmetric encryption is used to securely exchange a symmetric key, which is then used to encrypt the actual data. This is more efficient because symmetric encryption is faster.
Example:

- SSL/TLS: The client and server exchange symmetric encryption keys using asymmetric encryption to establish a secure session for data transfer.

BACKGROUND THEORY

RSA ALGORITHM

INTRODUCTION

The pioneering paper by Diffic and Hellman introduced a new approach to cryptography and, in effect, challenged cryptologists to come up with a cryptographic algorithm that met the requirements for public-key systems. One of the first of the responses to the challenge was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978 .The Rivest-Shamir- Adleman (RSA) scheme has since that time reigned supreme as the only widely accepted and implemented general-purpose approach to public-key encryption.

The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and n − 1 for some n. We examine RSA in this section in some detail, beginning with an explanation of the algorithm. Then we examine some of the computational and cryptanalytical implications of RSA.

KEY CONCEPTS:

PUBLIC KEY:

Used for encrypting data or verifying signatures.

PRIVATE KEY:

Used for decrypting data or signing messages.

STEPS OF RSA ALGORITHM

Key Generation

Select p, q	p and q both prime
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e)=1; 1 < e < \phi(n)$
Calculate d	$d=e^{-1} \bmod \phi(n)$
Public key	$KU=\{e,n\}$
Private key	$KR=\{d,n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod n$

Decryption

Plaintext:	C
Ciphertext:	$M = C^d \pmod n$

RSA APPLICATIONS:

- Secure Communications: Used in SSL/TLS for establishing encrypted communication channels (e.g., HTTPS).
- Digital Signatures: Used for signing messages or documents to ensure integrity and authenticity.
- Email Encryption: Used for encrypting and signing emails (e.g., with PGP).
- Cryptocurrency: Many cryptocurrencies, like Bitcoin, use RSA for secure transactions and wallet generation.

ADVANTAGES:

- Security: The private key is never transmitted, only the public key is shared.
- Non-repudiation: Digital signatures provide proof of the origin and integrity of the message.
- Scalability: Asymmetric encryption is ideal for situations where parties don’t have a secure channel to exchange a symmetric key.

DISADVANTAGES:

- Performance: RSA encryption is slower compared to symmetric encryption algorithms (like AES), especially with large data.
- Key Size: To achieve strong security, RSA keys need to be very large (typically 2048 bits or more), which can impact performance.

RSA IN PRACTICE:

RSA is often used in a hybrid encryption scheme, where it is used to exchange a symmetric key, and then that symmetric key is used to encrypt the actual data. This combines the security of asymmetric encryption with the efficiency of symmetric encryption.

RSA IMPLEMENTATION

LIBRARIES NEEDED

```
#include <SDL.h>
#include <stdio.h>
#include <stdbool.h>
#include <SDL_ttf.h>
#include <math.h>
```

SDL library for visualization

For functions like printf

For boolean variables

SDL library for Fonts

For math functions like sqrt

DECLARED CONSTANTS

```
#define max_s 50
#define WINDOW_WIDTH 1000
#define WINDOW_HEIGHT 700
```

Maximum size of texts

Dimentions of the window created for SDL visualization

```
10  void Ascii(char text[max_s],int ascii[max_s]){
11      int i;
12      for (i=0;i<max_s;i++){
13          ascii[i]=(int)text[i];
14
15      }
16
17
18  }
19  void Inv_Ascii(int ascii[max_s],char ciphertext[max_s]){
20  int i;
21  for (i=0;i<max_s;i++){
22      ciphertext[i]=(char)ascii[i];
23  }
24  }
```

Two functions that have inverse roles
Ascii turns text into ASCII and Inv_Ascii turns Ascii into text

```c
int fact_totient(int E, int T){
    if (T%E==0){
        return 0;
    }
    return 1;
}
```

Function to verify if the public key is a factor of the Euler totient(phi)

```c
int prime(int E){
    int i;
    if (E<=1){
        return 1;

    }
    for(i=2;i<=sqrt(E);i++){
        if (E%i==0){
            return 1;
        }
    }
    return 0;
}
```

Function to verify if the public key is prime

```c
/*function to check if the public key is valid
1* it has to be prime
2*it has to be less than the totient
3*it mustn't be a factor of the totient
*/
int public_key(int E,int T){
    if (prime(E)==0 && E<=T && fact_totient(E,T)==1){
        return 0;
    }
    return 1;

}
```

function that verifies if the public key is valid

Function for the key generation

```c
int Generating_keys(int q,int p,int*n,int*T){
    if(prime(p)==1 || prime(q)==1){
        printf("the number you just antered is not prime try another one ");
    }
    *n=q*p; //semi prime number//
    *T=(p-1)*(q-1);//this totient is especially for semi prime numbers such that n=p*q with q!=p and q&p are prime numbers //

}
```

Due to the difficulty to calculate the power of integers and modulos combined I created this function that has some optimizations which reduces the number of multiplications that have to be executed

```c
/*I created this version of the algorithm to calculate the modular exponentiation function
that already exists in the library math is not efficient when it comes to large numbers the algorithm that calculates it
is called exponentiation by squaring and it reduces the number of number of multiplications needed and does it in an logarithmic time
complexity*/
int modular_exponential_function (int Message_or_ciphertext, int K,int n){
    int mod=1;
    Message_or_ciphertext=Message_or_ciphertext%n;
    while (K>0){
        if (K%2==1){
            mod=(mod*Message_or_ciphertext)%n;
        }
        K=K/2;
        Message_or_ciphertext=(Message_or_ciphertext*Message_or_ciphertext)%n;

    }
    return mod;
}
```

Function for the encryption and the decryption using private key and public key

```c
void Encrypt(int ascii[max_s],int ciphertext[max_s],int E,int T,int n){
    int i;
    if (public_key(E,T)==0){

    for(i=0;i<max_s;i++){
        ciphertext[i]= modular_exponential_function(ascii[i],E,n);
    }
    }

}
```

```c
void decrypt(int ciphertext[max_s],int ascii_decrypted_text[max_s],int D,int n,int T,int E){
    int i;
    if( private_key(E,T,D)==0){
        for(i=0;i<max_s;i++){
            ascii_decrypted_text[i]=modular_exponential_function(ciphertext[i],D,n);
        }
    }
}
```

Function to verify if the private key is valid

```c
int private_key(int E, int T,int D){
    if ((D*E)%T==1){
        return 0;
    }
    return 1;
}
```

main Function

```c
int main(int argc, char* argv[]) {
    char text[max_s]="Hello world,";
    int i,n,T,q,p,E,D;
    q=61;
    p=53;
    E=17;
    D=2753;
    int ascii_of_text[max_s];
    int ciphertext[max_s];
    int ascii_decrypted_text[max_s];
    char decrypted_text[max_s];
```

this is for SDL visualization of the publik key and private key and the function sprintf turns the number into a text that can be printed in the window

```
char textE[max_s]="PUBLIC KEY :";
char textD[max_s]="PRIVATE KEY :";
char numberTextE[max_s];
char numberTextD[max_s];
char numciphertext[max_s];/*This is to use sprintf with a table not just a one value*/
char temp[max_s];
for(i=0;i<max_s;i++){
    sprintf(temp,"%d ",ciphertext[i]);
    strcat(numciphertext,temp);
}

sprintf(numberTextE, "E: %d", E);
sprintf(numberTextD, "D: %d", D);
```

initialization of SDL and TTF and window and renderer creation for the visualization with verification if the latter were created

```
/*initializing SDL*/
if (SDL_Init(SDL_INIT_VIDEO) < 0 || TTF_Init() < 0) {
    SDL_Log("SDL initialization failed: %s", SDL_GetError());
    return 1;
}
/*initializing ttf*/
 if (TTF_Init() < 0) {
    printf("TTF initialization failed: %s\n", TTF_GetError());
    SDL_Quit();
    return 1;
}
/*window creation*/
SDL_Window* window = SDL_CreateWindow("RAS visualization", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, 1000, 700, SDL_WINDOW_SHOWN);
if (!window) {
    SDL_Log("Window creation failed: %s", SDL_GetError());
    SDL_Quit();
    return 1;
}
/* Create a renderer*/
SDL_Renderer* renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
if (!renderer) {
    SDL_Log("Renderer creation failed: %s", SDL_GetError());
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 1;
}
```

Loading font using TTF_font function with verification

```
TTF_Font* font = TTF_OpenFont("C:\\Users\\mchel\\OneDrive\\Desktop\\encryption_algo_project-main\\hadjer\\font\\consolab.ttf", 100);
if (!font) {
    SDL_Log("Font loading failed: %s", TTF_GetError());
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    TTF_Quit();
    return 1;
    }
```

Creation of surfaces for texts using SDL_surface function from SDL library

```
    SDL_Color textColorE = {255, 255, 255, 255};
    SDL_Color textColorD = {255,255,255, 255};
    SDL_Surface* textSurfaceE = TTF_RenderText_Solid(font, numberTextE, textColorE);
    if (!textSurfaceE) {
        printf("Failed to create text surface E: %s\n", TTF_GetError());
        TTF_CloseFont(font);
        SDL_DestroyRenderer(renderer);
        SDL_DestroyWindow(window);
        TTF_Quit();
        SDL_Quit();
        return 1;
    }
    SDL_Surface* textSurfaceD = TTF_RenderText_Solid(font, numberTextD, textColorD);
    if (!textSurfaceD) {
        printf("Failed to create text surface D: %s\n", TTF_GetError());
        SDL_FreeSurface(textSurfaceE);
        TTF_CloseFont(font);
        SDL_DestroyRenderer(renderer);
        SDL_DestroyWindow(window);
        TTF_Quit();
        SDL_Quit();
        return 1;
    }
```

Creating Textures for texts using SDL_Texture function

```
  /*texture for cipher*/
SDL_Texture* cipherTexture = SDL_CreateTextureFromSurface(renderer, textSurfaceCipher);
SDL_FreeSurface(textSurfaceCipher); // Free the surface after creating the texture
if (!cipherTexture) {
    SDL_Log("Texture creation failed: %s", SDL_GetError());
    TTF_CloseFont(font);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    TTF_Quit();
    return 1;
}


/*texture for E & D*/

SDL_Texture* textTextureE = SDL_CreateTextureFromSurface(renderer, textSurfaceE);
SDL_FreeSurface(textSurfaceE); // Free the surface after creating the texture
if (!textTextureE) {
    SDL_Log("Texture creation failed: %s", SDL_GetError());
    TTF_CloseFont(font);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    TTF_Quit();
    return 1;
}
```

Defining Rectangles for visualization

```
int speed = 5;
/*defining the rects*/
SDL_Rect rectE = {100, 100, 200, 100};
SDL_Rect rectD = {680, 100, 200, 100};
SDL_Rect recttext = {300,210,380,60};
SDL_Rect rectciphertext = {100,390,800,60};
SDL_Rect rectdecryptedtext = {300,590,380,60};
/*defining rects text*/
SDL_Rect Etext = {100, 40, 200, 70};
SDL_Rect Dtext = {680, 40, 200, 70};
SDL_Rect textRectE = {120, 110, 160, 80};
SDL_Rect textRectD = {700, 110, 160, 80};
SDL_Rect textRecttext = {325,217,330,50};
SDL_Rect textRectciphertext = {150,395,700,50};
SDL_Rect textdecypted = {325,597,330,50};
```

animation and creation of one of the keys resulting in the visualization

ANIMATION

```
int key1X = 100;
int key1Y = 330;


int key2X = 100;
int key2Y = 510;



    // Main loop
    bool quit = false;
    SDL_Event event;
    while (!quit) {
        while (SDL_PollEvent(&event)) {
            switch (event.type) {
                case SDL_QUIT:
                    quit = true;
                    break;
                default:
                    break;
            }
        }
```

CREATION

```
    /*E key*/

    key1X += speed;
    if (key1X > 1000) {
        key1X = -100; // Reset to the left of the screen
    }


    int centerX = key1X, centerY = key1Y, radius = 50;
    for (int w = 0; w < radius * 2; w++) {
        for (int h = 0; h < radius * 2; h++) {
            int dx = radius - w; // Horizontal distance from center
            int dy = radius - h; // Vertical distance from center
            if ((dx * dx + dy * dy) <= (radius * radius)) {
                SDL_SetRenderDrawColor(renderer, 255, 255, 0, 255);
                SDL_RenderDrawPoint(renderer, centerX + dx, centerY + dy);
            }
        }
    }


    SDL_Rect shaft = {centerX + radius, centerY - 10, 200, 20};
    SDL_SetRenderDrawColor(renderer, 255, 255, 0, 255);
    SDL_RenderFillRect(renderer, &shaft);

    SDL_Rect tooth1 = {centerX + radius + 50, centerY + 10, 20, 20};
    SDL_Rect tooth2 = {centerX + radius + 100, centerY + 10, 20, 20};
    SDL_SetRenderDrawColor(renderer, 255, 255, 0, 255);
    SDL_RenderFillRect(renderer, &tooth1);
    SDL_RenderFillRect(renderer, &tooth2);
```

```
SDL_Color textColorkeyhandle = {255, 255, 255, 255};
SDL_Surface* textSurfacekh = TTF_RenderText_Solid(font, "E", textColorkeyhandle);
if (textSurfacekh) {
    SDL_Texture* textTexturekh = SDL_CreateTextureFromSurface(renderer, textSurfacekh);
    SDL_FreeSurface(textSurfacekh);

    SDL_Rect textRect;
    textRect.x = centerX - radius / 2;
    textRect.y = centerY - radius / 2;
    textRect.w = radius;
    textRect.h = radius;

    SDL_RenderCopy(renderer, textTexturekh, NULL, &textRect);
    SDL_DestroyTexture(textTexturekh);
}
```

Coloration and placement of the texts in the suitable positions

```
/*Clear the screen*/
SDL_SetRenderDrawColor(renderer, 246, 36, 89,1);/*this is for background*/
SDL_RenderClear(renderer);


SDL_SetRenderDrawColor(renderer,128, 0, 128, 255);/*this is for public key AKA E*/
SDL_RenderFillRect(renderer, &rectE);


SDL_SetRenderDrawColor(renderer,128, 0, 128, 255);/*this is for the private key AKA D*/
SDL_RenderFillRect(renderer, &rectD);


SDL_SetRenderDrawColor(renderer,128, 0, 128, 255);/*this for the Encrypted text*/
SDL_RenderFillRect(renderer, &recttext);

SDL_SetRenderDrawColor(renderer,128, 0, 128, 255);/*this is for the Encription of the text using the public key E*/
SDL_RenderFillRect(renderer, &rectciphertext);

SDL_SetRenderDrawColor(renderer,128, 0, 128, 255);/*this for the decrypted text*/
SDL_RenderFillRect(renderer, &rectdecryptedtext );



SDL_RenderCopy(renderer,textTextureEtext,NULL,&Etext);
SDL_RenderCopy(renderer,textTextureDtext,NULL,&Dtext);
SDL_RenderCopy(renderer, textTextureE, NULL, &textRectE);/*this is for public key AKA E*/
SDL_RenderCopy(renderer, textTextureD, NULL, &textRectD);/*this is for the private key AKA D*/
SDL_RenderCopy(renderer, textTexturetext, NULL, &textRecttext);/*this for the Encrypted text*/
SDL_RenderCopy(renderer,cipherTexture,NULL,&textRectciphertext);/*this is for the Encription of the text using the public key E*/
SDL_RenderCopy(renderer, textTexturetext, NULL, &textdecypted);/*this for the Encrypted text*/
```

setting the right delay for the animation and the window presentation and cleaning up SDL and TTF structures

```
        /*Present the renderer*/
        SDL_RenderPresent(renderer);


        // Delay to control
        SDL_Delay(50);
    }


    /*Cleanup and clear*/
    SDL_DestroyTexture(textTexturedecrypted);
    SDL_DestroyTexture(textTextureEtext);
    SDL_DestroyTexture(textTextureDtext);
    SDL_DestroyTexture(cipherTexture);
    SDL_DestroyTexture(textTexturetext);
    SDL_DestroyTexture(textTextureE);
    SDL_DestroyTexture(textTextureD);
    TTF_CloseFont(font);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    TTF_Quit();
    SDL_Quit();


    return 0;
}
```

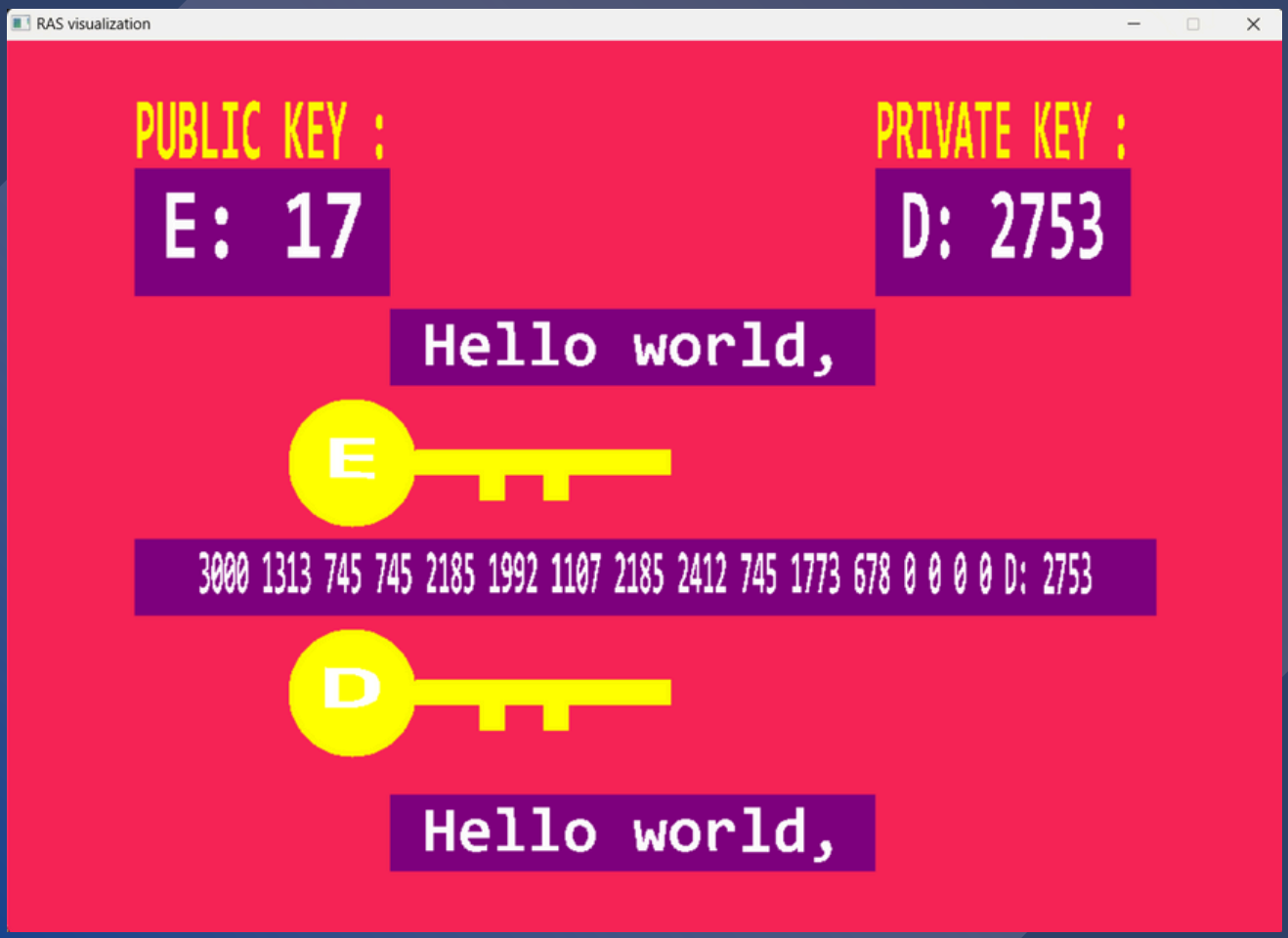
RSA CODE OUTPUT

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

the text i want to Encrypt
Hello world,

the ASCII of text to be Encrypted
721011081081113211911111410810044000000000000000000000000000000000000000
Ciphertext:
300013137457452185199211072185241274517736780000000000000000000000000000000000
ASCII of Dercypted text
721011081081113211911111410810044000000000000000000000000000000000000000
The Decrypted text:
Hello world,
```

SDL WINDOW



GOLDWASSER-MICALI

INTRODUCTION

The Goldwasser-Micali (GM) cryptosystem, introduced by Shafi Goldwasser and Silvio Micali in 1982, is one of the most significant contributions to modern cryptography. It provided a groundbreaking method of encrypting data, laying the foundation for probabilistic encryption and achieving security levels previously thought unattainable. This system not only changed the way we approach encryption but also influenced many areas in cryptographic research.

KEY CONCEPT:

- Probabilistic Encryption
- Before the GM cryptosystem, encryption methods were largely deterministic—encrypting the same message with the same key always produced the same ciphertext. Goldwasser and Micali introduced the idea of probabilistic encryption, where the same plaintext can result in different ciphertexts. This innovation significantly improved security by making encrypted messages harder to analyze through patterns.Semantic Security
- The GM cryptosystem is semantically secure, meaning it ensures that an attacker cannot deduce any information about the plaintext from the ciphertext. This property guarantees the confidentiality of encrypted data, even in the face of powerful adversaries.Mathematical Foundation

The system relies on the quadratic residuosity problem, a well-known hard problem in number theory. This problem involves determining whether a given number is a quadratic residue modulo a composite number, which is computationally infeasible without specific knowledge (like the factorization of the composite).

HOW THE GM CRYPTOSYSTEM WORKS

- The Goldwasser-Micali cryptosystem uses a public-private key pair for encryption. The private key consists of two large prime numbers, and the public key includes the modulus (their product) and a quadratic non-residue modulo the modulus. To encrypt a bit, a random number is chosen, and the ciphertext is computed in a way that ensures different ciphertexts for the same plaintext. Decryption checks if the ciphertext is a quadratic residue modulo the modulus; if it is, the bit is zero; if not, the bit is one.
- The system's probabilistic nature provides strong security, as even identical messages produce different ciphertexts. It's based on simple modular arithmetic but forms the basis for advanced cryptographic techniques like secure multi-party computation, zero-knowledge proofs, and homomorphic encryption.
-

CHALLENGES AND LIMITATIONS

- The Goldwasser-Micali cryptosystem uses a public key that includes a modulus and a quadratic non-residue, and a private key based on two large primes. Encryption involves choosing a random number, ensuring different ciphertexts for the same bit. Decryption checks if the ciphertext is a quadratic residue; if it is, the bit is zero, otherwise, it's one.
- This system's probabilistic nature provides strong security and is foundational for advanced cryptographic techniques like secure multi-party computation, zero-knowledge proofs, and homomorphic encryption.

DIFFIE_HELLMAN ALGORITHM

INTRODUCTION

The Diffie-Hellman algorithm is a cryptographic protocol used for secure key exchange over an insecure communication channel. It was introduced in 1976 by Whitfield Diffie and Martin Hellman, marking a significant breakthrough in public-key cryptography. The primary goal of the algorithm is to allow two parties to generate a shared secret key that can be used for encryption and decryption, without directly transmitting the key itself.

KEY FEATURES:

1. Purpose: Securely establish a shared secret key between two parties without prior shared secrets.
2. Type: Asymmetric cryptographic method used primarily for key exchange.
3. Foundation: Based on the mathematical properties of modular arithmetic and the discrete logarithm problem.

SECURITY:

The security of the Diffie-Hellman algorithm relies on the difficulty of the discrete logarithm problem.

APPLICATIONS:

- Secure Communication: Establishing a shared key for symmetric encryption (e.g., AES).
- TLS/SSL Protocols: Used in HTTPS for secure connections.
- VPNs: Ensuring secure communication between endpoints.

LIMITATIONS:

1. Man-in-the-Middle Attack: Diffie-Hellman on its own does not authenticate the communicating parties. It must be combined with authentication mechanisms like digital signatures or certificates.
2. Group Selection: Weak parameters can compromise security.

VARIANTS:

- Elliptic Curve Diffie-Hellman (ECDH): A variant of Diffie-Hellman that operates over elliptic curves, offering stronger security with smaller key sizes.

The Diffie-Hellman algorithm remains a cornerstone of modern cryptography and is widely used for secure key exchange in various protocols and systems.

DIFFIE HELLMAN IMPLEMENTATION

Code Breakdown: Function to Calculate Modular Exponentiation

```
// Function to calculate (base^exponent) % modulus
int calculate_modulo(int base, int exponent, int modulus) {
    int result = 1;
    for (int i = 0; i < exponent; i++) {
        result = (result * base) % modulus; // Multiply and take remainder
    }
    return result;
}
```

The function computes modular exponentiation,
finding the remainder of a base raised to a power divided by a modulus
for example (2^3 mod 5 = 3).
It is essential for generating public keys and shared secrets in the Hellman Key Exchange

Calculating Public Keys

```
publicKeyAlice = calculate_modulo(G, privateKeyAlice, P);
publicKeyBob = calculate_modulo(G, privateKeyBob, P);
```

Purpose: Use modular exponentiation to compute public keys.
Alice's Public Key:
Bob's Public Key:
Where is the primitive root, is the prime modulus, and the private keys are Alice's and Bob's secret values.

Calculating Shared Secret Keys

```
sharedSecretAlice = calculate_modulo(publicKeyBob, privateKeyAlice, P);
sharedSecretBob = calculate_modulo(publicKeyAlice, privateKeyBob, P);
```

Purpose: Derive the shared secret key using exchanged public keys and private keys.
Alice's Shared Secret:
Bob's Shared Secret:
Result: Both shared secrets must be identical, ensuring secure communication

Input Values for Diffie-Hellman

```
printf("Enter a prime number (P): ");
scanf("%d", &P);
printf("Enter a primitive root (G): ");
scanf("%d", &G);

printf("Enter Alice's private key: ");
scanf("%d", &privateKeyAlice);
printf("Enter Bob's private key: ");
scanf("%d", &privateKeyBob);
```

Prime Modulus ( ): A large prime number for the calculations.
Primitive Root ( ): A number whose powers generate distinct values modulo .
Private Keys: Secret keys chosen by Alice and Bob to compute public keys.

Code Execution Result

```
C:\Users\Surface\Downloads\
Enter a prime number (P): 3
Enter a primitive root (G): 2
Enter Alice's private key: 4
Enter Bob's private key: 5
Alice's Public Key: 1
Bob's Public Key: 2
Shared Secret Key (Alice): 1
Shared Secret Key (Bob): 1
Key exchange successful! Shared secret key is: 1

Process returned 0 (0x0)   execution time : 19.166 s
Press any key to continue.
```

Visualization of Hellman key exchange Algorithm

1/ Importing libraries

```c
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <stdio.h>
#include <stdbool.h>

// Screen dimensions
const int SCREEN_WIDTH = 800;
const int SCREEN_HEIGHT = 600;
```

SDL2 Libraries: SDL2/SDL.h and SDL2/SDL_ttf.h are used for creating the window , rendering graphics, and displaying text

Standard Libraries: stdio.h for input/output and stdbool.h for boolean logic.

hese setup basics are essential for initializing the graphical environment.

2/ SDL libraries initialization

```c
int main() {
    // Initialize SDL
    if (SDL_Init(SDL_INIT_VIDEO) != 0) {
        printf("Error initializing SDL: %s\n", SDL_GetError());
        return 1;
    }

    // Initialize SDL_ttf
    if (TTF_Init() != 0) {
        printf("Error initializing SDL_ttf: %s\n", TTF_GetError());
        SDL_Quit();
        return 1;
    }
```

1. SDL_Init: Initializes the SDL library to handle video and events.
2. TTF_Init: Enables font rendering for adding text labels.
3. Error Handling: Prints an error message if initialization fails.

3/ Creating the Window and Renderer

```
SDL_Window *window = SDL_CreateWindow("Diffie-Hellman Visualization",
                                      SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
                                      SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_SHOWN);
if (!window) {
    printf("Error creating window: %s\n", SDL_GetError());
    TTF_Quit();
    SDL_Quit();
    return 1;
```

1. SDL_CreateWindow: Creates a window with dimensions 800x600 for the visualization.
2. SDL_CreateRenderer: Sets up an accelerated renderer for drawing graphics efficiently.

4/ Drawing Rectangles and Rendering Text

```
// Function to draw a filled rectangle
void draw_rectangle(SDL_Renderer *renderer, int x, int y, int w, int h, SDL_Color color) {
    SDL_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);
    SDL_Rect rect = {x, y, w, h};
    SDL_RenderFillRect(renderer, &rect);
}

// Function to render text
void render_text(SDL_Renderer *renderer, TTF_Font *font, const char *text, int x, int y, SDL_Color color) {
    SDL_Surface *surface = TTF_RenderText_Solid(font, text, color);
    SDL_Texture *texture = SDL_CreateTextureFromSurface(renderer, surface);

    SDL_Rect rect = {x, y, surface->w, surface->h};
    SDL_RenderCopy(renderer, texture, NULL, &rect);

    SDL_FreeSurface(surface);
    SDL_DestroyTexture(texture);
}
```

1. draw_rectangle: Renders filled rectangles to represent Alice, Bob, and the shared secret.
2. render_text: Displays labels (e.g., "Common", "Secret") for clarity.

5/ Some Animation Logic

```
// Update rectangle positions (move Alice and Bob closer to the center)
if (alice_x < shared_secret_x - 50) alice_x += speed;
if (bob_x > shared_secret_x + 50) bob_x -= speed;
```

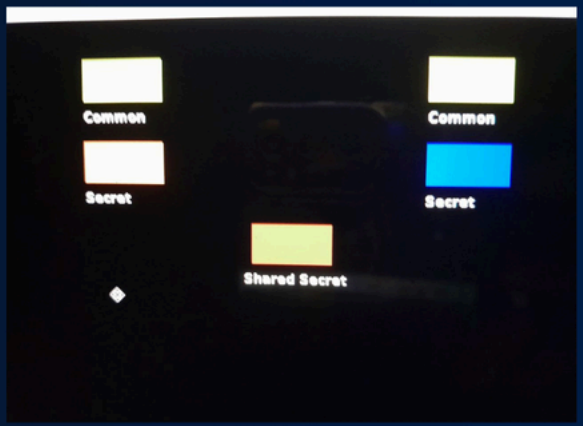
- Alice moves right (alice_x += speed) until she's near the shared secret.
- Bob moves left (bob_x -= speed) to do the same.
- It visually simulates the key exchange process.

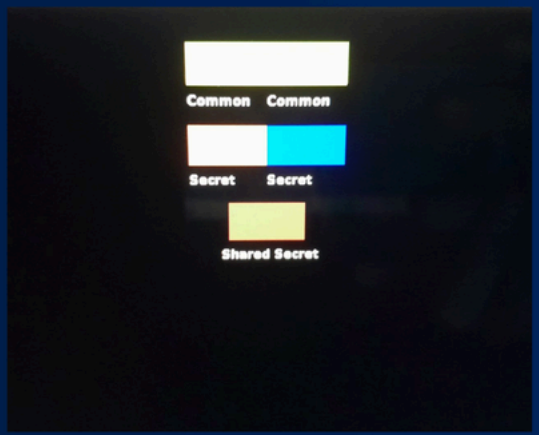
6/ Final Output and Cleanup

```
// Cleanup
TTF_CloseFont(font);
SDL_DestroyRenderer(renderer);
SDL_DestroyWindow(window);
TTF_Quit();
SDL_Quit();
```

1. Final Visual: Displays "Shared Secret" to indicate successful key exchange.
2. Resource Cleanup: Frees memory and shuts down SDL to ensure efficiency

Visualization Result





METHODOLOGY

IMPLEMENTATION

- Programming Language: C (with SDL for visualization).
- Libraries Used: SDL2 for graphics, GMP for handling large integers.
- Structure: Each algorithm implemented in modular functions for key generation, encryption, and decryption.

VISUALIZATION

- Purpose: To provide a graphical understanding of encryption and decryption processes.
- Tools: SDL for rendering.
- Design: Interactive user interface showing key generation, data encryption, and decryption in real-time.

CONCLUSION

Implementation Programming Language: C (with SDL for visualization). Libraries Used: SDL2 for graphics, GMP for handling large integers. Structure: Each algorithm implemented in modular functions for key generation, encryption, and decryption. Visualization Purpose: To provide a graphical understanding of encryption and decryption processes. Tools: SDL for rendering. Design: Interactive user interface showing key generation, data encryption, and decryption in real-time.

REFERENCES

Schneier, B. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C

First Steps in SDL Game Development -- Kameron Hussain & Frahaan Hussain -- 2024 -- Sonar Publishing

SDL Game Development_ Discover how to leverage the power of -- Shaun Mitchell -- 2013 -- Packt Publishing

Cryptography and Network Security_ Principles and Practice - -- Willian Stalings

Cryptography in C and C++ -- Michael Welschenbach -- Springer Nature, Berkeley, CA