Quebra RSA

Introdução

Este projeto implementa o algoritmo **RSA**, um dos métodos mais conhecidos de criptografia assimétrica.
A proposta do trabalho é, dado um texto cifrado com o **RSA** e sabendo que p e q são números primos menores que **1024(10 bits)** , criar um programa que encontre a chave privada d sendo que a chave pública
e, n é conhecida.
O programa criado foi feito da seguinte maneira:

1. Fatora o módulo público n para encontrar os primos p e q.
2. Calcula o totiente de Euler $\phi(n)$.
3. Determina a chave privada d a partir da chave pública
   e, n.
4. Cifra e decifra mensagens utilizando as chaves geradas.

O projeto inclui diversas funções para realizar essas operações, como geração de primos, cálculo do inverso modular, cifragem e decifragem de textos, além de uma interface interativa com o usuário para testar o funcionamento do sistema.

Índice

Função `sieve_of_eratosthenes`

A função `sieve_of_eratosthenes` é utilizada para gerar uma lista de números primos menores que um limite especificado. Ela implementa o Crivo de Eratóstenes, um algoritmo eficiente para encontrar todos os números primos em um intervalo.
Ler mais

Função `extended_gcd`

A função `extended_gcd` implementa o Algoritmo de Euclides Estendido para encontrar o Máximo Divisor Comum (MDC) de dois números e os coeficientes que satisfazem a equação linear correspondente.
Ler mais

Função `mod_inverse`

A função `mod_inverse` é utilizada para encontrar o inverso modular de e em relação a ϕ. Este cálculo é essencial para determinar a chave privada d no algoritmo RSA.
Ler mais

Função `find_private_key`

A função `find_private_key` tenta fatorar n para encontrar os valores de p e q, e utiliza o totiente de Euler para calcular a chave privada d.
Ler mais

Função `encrypt_plaintext`

A função `encrypt_plaintext` realiza a cifragem de um texto simples utilizando a chave pública e, n.
Ler mais

Função `decrypt_ciphertext`

A função `decrypt_ciphertext` realiza a decifragem de um texto cifrado utilizando a chave privada d e o módulo n.
Ler mais

Função `main`

A função `main` é o ponto de entrada do programa, oferecendo uma interface interativa para o

usuário inserir valores e realizar operações de cifragem e decifragem com o RSA.
Ler mais

Diretório `utils/`

A pasta `utils/` inclui arquivos auxiliares utilizados para testes e exemplos no contexto deste projeto. Estes arquivos permitem entender melhor o funcionamento do sistema RSA implementado.

Estrutura da Pasta

- `exemplos.txt` : Contém exemplos de entrada para e, n. Estes exemplos podem ser utilizados como base para testar a cifragem e decifragem de mensagens.
- `texto.txt` : Contém um texto usado como exemplo para ser cifrado. Este arquivo permite ao usuário ter um texto simples a disposição para ser usado como mensagem cifrada no algoritmo RSA.

Utilização

- `exemplos.txt` : Abra o arquivo para consultar exemplos pré-definidos de chaves públicas e, n:

```
(e, n)= (65537, 3233)
    e = 65537
    n = p × q = 61 × 53 = 3233

(e, n) = (7, 2773)
    e = 7
    n = p × q = 59 × 47 = 2773
    ....
```

- `texto.txt` : Use o texto contido neste arquivo como entrada para as opções **2** e **3**(Cifra e Decifra, respectivamente):

```
Lorem ipsum mauris augue morbi condimentum sollicitudin curabitur fusce luctus, elem
...
```

Voltar ao índice

sieve_of_eratosthenes(limit)

A função `sieve_of_eratosthenes` é utilizada para gerar uma lista de **números primos** menores que um limite especificado. Ela implementa o **Crivo de Eratóstenes**, um algoritmo eficiente para encontrar todos os números primos em um intervalo.

Parâmetros

- `limit` **(Limite):**
  - Um valor inteiro que representa o limite superior (exclusivo) para os números primos a serem encontrados.
  - A função gera todos os números primos menores que esse valor.

Funcionamento

1. **Inicialização:**
   - Cria uma lista chamada `sieve` onde cada índice representa um número inteiro.
   - Inicialmente, assume que todos os números são primos (“True”), exceto 0 e 1, que são marcados como não primos (“False”).
2. **Processamento:**
   - Itera sobre os números de 2 até a raiz quadrada de `limit` .
   - Para cada número primo encontrado, marca todos os seus múltiplos como não primos (“False”).
3. **Coleta de Primos:**
   - Retorna uma lista de números inteiros onde o valor correspondente em `sieve` é “True” (ou seja, os números primos).

Retorno

A função retorna uma lista contendo todos os números primos menores que o valor de `limit` .

Exemplo de Uso

Entradas:

- $\text{limit} = 10$

Saída:

A função retorna a lista:

```
[2, 3, 5, 7]
```

O Crivo de Eratóstenes é mais eficiente do que verificações diretas de primalidade para grandes conjuntos de números.

Voltar ao índice

extended_gcd(a, b)

A função `extended_gcd` implementa o **Algoritmo de Euclides Estendido** para encontrar o **Máximo Divisor Comum/Greatest Common Divisor(MDC/GCD)** de dois números e os coeficientes que satisfazem a seguinte equação linear:

$$a \cdot x + b \cdot y = \gcd(a, b)$$

Parâmetros

- **`a` (Inteiro):** Primeiro número inteiro.
- **`b` (Inteiro):** Segundo número inteiro.

Retorno

A função retorna uma tupla $(\gcd, x, y)$, onde:

- **`gcd`** : O máximo divisor comum entre a e b.
- **`x`** e **`y`** : Os coeficientes inteiros que satisfazem a equação $a \cdot x + b \cdot y = \gcd(a, b)$.

Funcionamento

1. **Caso Base:**
   - Se $b = 0$, a função retorna $(a, 1, 0)$, indicando que o MDC é a e os coeficientes são $x = 1$ e $y = 0$.
2. **Recursão:**
   - Caso contrário, a função faz uma chamada recursiva com $(b, a \bmod b)$.
   - Calcula os coeficientes x e y com base nos valores retornados pela chamada recursiva.

Exemplo de Uso

Entradas:

- $a = 30$
- $b = 12$

Saída:

A função retorna:

```
(6, -1, 3)
```

Isso significa que:

$$30 \cdot (-1) + 12 \cdot 3 = 6$$

Utilidade

- **Cálculo do Inverso Modular:** Encontrar o valor de x tal que $a \cdot x \equiv 1 \pmod{b}$.

Voltar ao índice

mod_inverse(e, phi)

A função `mod_inverse` é utilizada para encontrar o **inverso modular** de e em relação a ϕ. Este é um conceito fundamental na criptografia RSA, onde ele é usado para calcular a chave privada d. Sem o inverso modular, não seria possível determinar d, tornando inviável a decriptação no RSA.

Em termos matemáticos, a função encontra d tal que:

$$e \cdot d \equiv 1 \pmod{\phi(n)}$$

Parâmetros

- `e` **(Expoente público):**
  - Um valor inteiro utilizado como parte da chave pública no RSA.
  - Deve ser **coprimo** a $\phi(n)$, ou seja, o MDC/GCD("Máximo Divisor Comum/Greatest Common Divisor") entre e e $\phi(n)$ é igual a 1 ($\gcd(e, \phi) = 1$).
- `phi` **(Totiente de Euler):**
  - Representa $\phi(n)$, que é o produto das reduções de cada fator primo de n. Para $n = p \cdot q$, onde p e q são primos, temos:
    $\phi(n) = (p - 1) \cdot (q - 1)$
  - Esse valor é essencial para calcular a chave privada.

Retorno

A função retorna o valor do inverso modular d, ajustado para o intervalo $[0, \phi - 1]$, se e e ϕ forem coprimos.
Se o inverso modular não existir (ou seja, se $\gcd(e, \phi) \neq 1$), a função lança uma exceção com a mensagem:

```
Inverso modular não existe.
```

Exemplo de Uso

Entradas:

- $e = 7$
- $\phi = 40$

Saída:

A função retorna $d = 23$, pois:

$$7 \cdot 23 \equiv 1 \pmod{40}$$

Voltar ao índice

find_private_key(e, n)

A função `find_private_key` é usada para calcular a **chave privada** d no algoritmo RSA. Dado o expoente público e e o módulo n (produto de dois números primos p e q), a função tenta fatorar n para encontrar os valores de p e q, e então calcula $\phi(n)$ e d.

Parâmetros

- `e` **(Expoente público):**
  - Um inteiro usado na chave pública no RSA.
- `n` **(Módulo):**
  - O produto de dois números primos p e q.

Retorno

A função retorna uma tupla (d, p, q), onde:

- `d` : A chave privada calculada usando o inverso modular de e em relação a $\phi(n)$.
- `p` e `q` : Os fatores primos de n.

Funcionamento

1. **Geração de Primos:**
   - Usa a função sieve_of_eratosthenes para gerar uma lista de números primos até 1024.
2. **Fatorização de n:**
   - Tenta encontrar dois fatores primos p e q tais que $n = p \cdot q$.
   - Itera sobre a lista de números primos gerados. Para cada primo:
     - Verifica se $n \mod \text{primo} = 0$, indicando que o primo é um fator de n.
     - Calcula $q = n/p$ e verifica se q também é primo.
3. **Erro de Fatorização:**
   - Se p ou q não forem encontrados, a função lança uma exceção.

4. **Cálculo de $\phi(n)$:**
   - Usa a fórmula $\phi(n) = (p-1) \cdot (q-1)$.
5. **Cálculo de d:**
   - Usa a função `mod_inverse` para encontrar d, o inverso modular de e em relação a $\phi(n)$.

Exemplo de Uso

Entradas:

- $e = 7$
- $n = 55$ (com os fatores primos $p = 5$, $q = 11$)

Saída:

A função retorna:

```
(23, 5, 11)
```

Isso significa que:

- $d = 23$ é a chave privada.
- $p = 5, q = 11$.

Observações

- Esta implementação usa um limite para os números primos (1024), o que limita sua aplicação prática para n maiores.
- Em sistemas reais, n é gerado usando primos muito maiores, tornando a fatorização computacionalmente inviável, logo mais **seguro**.

Voltar ao índice

encrypt_plaintext(plaintext, e, n)

A função `encrypt_plaintext` é usada para cifrar um texto simples (“plaintext”) utilizando a **chave pública**
e, n com o algoritmo RSA. A cifragem transforma cada caracter do texto em um valor numérico cifrado baseado na fórmula:

$$c \equiv m^e \pmod{n}$$

onde:

- c é o caractere cifrado.
- m é o valor numérico do caractere original (usando o código ASCII).
- e é o expoente público da chave.
- n é o módulo da chave.

Parâmetros

- `plaintext` **(Texto Simples):**
  - Uma string contendo o texto a ser cifrado.
- `e` **(Expoente Público):**
  - O valor inteiro da chave pública.
- `n` **(Módulo):**
  - O valor inteiro do módulo da chave pública.

Retorno

A função retorna uma string contendo os valores cifrados separados por espaços, onde cada valor representa um caractere do texto original cifrado.

Funcionamento

1. **Iteração sobre o Texto:**
   - Para cada caractere do texto de entrada (`plaintext`), converte o caractere em seu valor ASCII usando `ord(char)` .
2. **Cálculo da Cifra:**
   - Aplica a fórmula $c = m^e \mod n$ para cifrar o valor ASCII.
3. **Agregação dos Valores Cifrados:**
   - Converte cada valor cifrado em uma string e junta os valores separados por espaços.

Exemplo de Uso

Entradas:

- `plaintext` **:** "ABC"
- `e` **:** 3
- `n` **:** 33

Saída:

A função retorna:

```
8 1 27
```

Explicação:

1. Para o caractere “A” (ASCII 65):

$$c = 65^3 \bmod 33 = 8$$

2. Para o caractere “B” (ASCII 66):

$$c = 66^3 \bmod 33 = 1$$

3. Para o caractere “C” (ASCII 67):

$$c = 67^3 \bmod 33 = 27$$

Observações

- O tamanho do texto cifrado dependerá de n, pois valores maiores de n permitem cifrar caracteres ASCII mais complexos.

Voltar ao índice

decrypt_ciphertext(ciphertext, d, n)

A função `decrypt_ciphertext` é usada para decifrar um texto cifrado utilizando a **chave privada** d e o módulo n, com o algoritmo RSA. A decifra transforma cada bloco cifrado em seu caractere original baseado na fórmula:

$$m \equiv c^d \pmod{n}$$

onde:

- m é o valor decifrado do caractere original.
- c é o valor numérico cifrado (bloco cifrado).
- d é o expoente privado da chave.
- n é o módulo da chave.

Parâmetros

- **`ciphertext` (Texto Cifrado):**
  - Uma string contendo os blocos cifrados separados por espaços.
- **`d` (Chave Privada):**

    - O valor inteiro da chave privada.
- **`n` (Módulo):**
    - O valor inteiro do módulo da chave privada.

Retorno

A função retorna o texto decifrado como uma string, reconstruindo os caracteres originais a partir dos blocos cifrados.

Funcionamento

1. **Divisão dos Blocos:**
    - Divide o texto cifrado em blocos utilizando o espaço como delimitador.
2. **Decifração de Cada Bloco:**
    - Converte cada bloco em um inteiro.
    - Aplica a fórmula $m = c^d \mod n$ para calcular o valor decifrado.
    - Converte o valor decifrado de volta para um caractere ASCII usando `chr()`.
3. **Validação:**
    - Verifica se cada bloco cifrado c é menor que n. Caso contrário, uma exceção é lançada indicando um erro na cifra.
4. **Agregação dos caracteres:**
    - Junta todos os caracteres decifrados para reconstruir o texto original.

Exemplo de Uso

Entradas:

- **`ciphertext`:** "8 1 27"
- **`d`:** 3
- **`n`:** 33

Saída:

A função retorna:

```
ABC
```

Explicação:

1. Para o bloco cifrado $c = 8$:
   $m = 8^3 \mod 33 = 65 \quad (\text{caractere: “A”})$
2. Para o bloco cifrado $c = 1$:
   $m = 1^3 \mod 33 = 66 \quad (\text{caractere: “B”})$

3. Para o bloco cifrado $c = 2$:
   $m = 27^3 \mod 33 = 67 \quad (\text{caractere: “C”})$

Observações

- A função pressuppõe que os blocos cifrados foram gerados corretamente com uma chave pública válida.
- É necessário garantir que os valores de d e n correspondam à chave privada correta para o texto cifrado, caso contrário não funcionará.

Voltar ao índice

main()

A função `main` é o ponto de entrada do programa para cifragem e decifragem utilizando o algoritmo RSA. Ela permite que o usuário interaja com o sistema, forneça chaves públicas, realize operações de cifragem e decifragem, e visualize os resultados.

Funcionamento

1. **Entrada de Dados:**
   - Solicita ao usuário os valores de e (expoente público) e n (produto dos fatores primos p e q).
   - Valida os dados de entrada para garantir que sejam inteiros.
2. **Cálculo da Chave Privada:**
   - Utiliza a função `find_private_key` para calcular a chave privada d, bem como os fatores p e q.
   - Em caso de erro na fatorização ou no cálculo, exibe uma mensagem de erro.
3. **Menu de Operações:**
   - Exibe um menu com as opções:
     - **1:** Cifrar um texto.
     - **2:** Decifrar um texto cifrado.
     - **3:** Sair do programa.
   - Solicita ao usuário que escolha uma opção e realiza a ação correspondente.

   3.1. **Cifrar um Texto:**
   - Solicita ao usuário um texto simples para ser cifrado.
   - Usa a função `encrypt_plaintext` para realizar a cifragem com os valores de e, n
   - Exibe o texto cifrado como uma sequência de blocos numéricos separados por espaços.

3.2. **Decifrar um Texto Cifrado:**

- Solicita ao usuário um texto cifrado (sequência de números separados por espaços).
- Usa a função `decrypt_ciphertext` para realizar a decifragem com os valores de d e n.
- Exibe o texto decifrado como uma string legível.

4. **Encerramento:**
   - O programa continua executando até que o usuário escolha a opção de sair.

Exemplo de Uso

Fluxo:

1. O usuário insere os valores:
   - $e = 7$
   - $n = 55$
2. O programa calcula:
   - $d = 23$
   - $p = 5, q = 11$
3. O usuário escolhe:
   - **Opção 1 (Cifrar):**
     - Texto: `ABC`
     - Saída: `8 1 27`
   - **Opção 2 (Decifrar):**
     - Texto: `8 1 27`
     - Saída: `ABC`
4. O usuário escolhe a opção **3 (Sair)** e o programa encerra.

Observações

- A função é projetada para fins didáticos e não é adequada para uso em aplicações reais devido à limitação no tamanho das chaves e falta de otimizações.

Voltar ao índice