Entrega Final - Jogo utilizando criptografia RSA e comunicação de rede

José Heitor Pagotto RA 12225006

Projeto desenvolvido em dupla com:
Wesley Lencione de Oliveira RA 12225140

1. Tema

O objetivo do projeto é criar uma versão do **jogo da velha** em python, que utiliza o método de criptografia RSA para segurança de dados, e se comunica com outra instância de aplicação através de sockets TCP.

2. Funcionamento do Jogo

O jogo irá funcionar alternando entre os jogadores, onde cada um escolherá um número de 0 a 8, correspondentes às posições do tabuleiro, preenchendo os campos com os valores “X” ou “O” até que seja feita uma sequência de três símbolos seguidos, ou que acabe em empate.

3. Protocolo de Troca de Mensagens

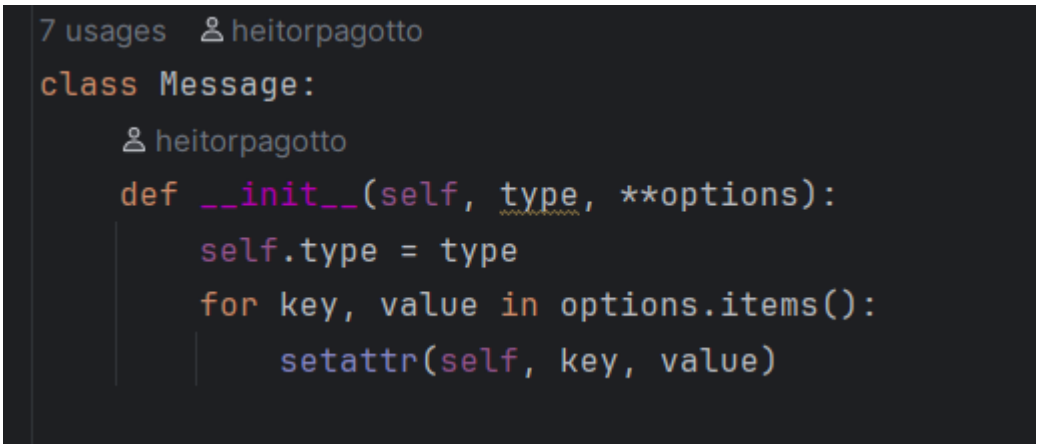
Para a troca de mensagens será aberto um socket TCP em uma porta configurável, que irá se conectar com outra instância da aplicação, onde será feito o envio e recebimento de dados para realizar as jogadas.

As mensagens serão **JSONs** com informações sobre a jogada atual:

Tipo	Origem -> Destino	Formato	Descrição
KEY_EXCH	Cliente -> Servidor	`{ “type”: “KEY_EXCH”, “n”:”<base10>”, “e”: “<base10>” }`	Envio de chave pública RSA
KEY_ACK	Ambos	`{ “type”: “KEY_ACK” }`	Confirmação de recebimento da chave pública
START	Primeiro a iniciar	`{ “type”: “START” }`	Início do jogo

MOVE	Alternado	{ "type": "MOVE", "data": "<C>" }	Jogada cifrada
END	Declarador do fim	{ "type": "END", "result": "X" }	Fim do jogo

4. **Estrutura de dados da Mensagem**
   Para o envio de mensagens a estrutura definida é uma classe com o tipo definido como string fixa, enquanto as outras propriedades virão dinamicamente de acordo com o que for informado no construtor



5. **Funcionamento do Algoritmo**
   a. **RSA**
      O código para criptografia RSA está em uma classe chamada `RsaCryptography`.
      Na inicialização (`__init__`), o construtor aceita opcionalmente `n`, `e` e `d`. Se `n` e `d` forem fornecidos, ele configura a chave privada. Caso ambas as listas fiquem vazias, chama `_calculate_rsa_variables()` para gerar novos pares de chaves a cada instância da classe.

```python
class RsaCryptography:
    _privateKey = []
    _publicKey = []

    heitorpagotto
    def __init__(self, n=None, e=None, d=None):
        if n is not None and d is not None:
            self._privateKey = [n, d]
        else:
            self._privateKey = []

        if n is not None and e is not None:
            self._publicKey = [n, e]
        else:
            self._publicKey = []

        if len(self._privateKey) == 0 and len(self._publicKey) == 0:
            self._calculate_rsa_variables()
```

O método `_calculate_rsa_variables()` escolhe aleatoriamente dois primos `p` e `q` no intervalo de 0 a 100, calcula `n = p * q` e φ(n) (`delta = (p-1)*(q-1)`), depois seleciona um `e` coprimo com φ(n) e calcula a modular inversa `d`, gerando um par de chaves.

```python
def _calculate_rsa_variables(self):
    min_random_prime = 0
    max_random_prime = 100

    initial_primes = self._generate_prime_numbers(min_random_prime, max_random_prime)

    p = initial_primes[random.randint( a: 0, len(initial_primes) - 1)]

    initial_primes = self._generate_prime_numbers(p + 1, max_random_prime)

    q = initial_primes[random.randint( a: 0, len(initial_primes) - 1)]

    while q == p:
        q = initial_primes[random.randint( a: 0, len(initial_primes) - 1)]

    n = p * q
    delta = (p - 1) * (q - 1)

    initial_primes = self._generate_prime_numbers(q + 1, delta)

    e = initial_primes[random.randint( a: 0, len(initial_primes) - 1)]
    d = self._mod_inverse(e, delta)

    self._privateKey = [n, d]
    self._publicKey = [n, e]
```

Por fim, a criptografia converte cada caractere em seu código Unicode e aplica a operação modular. A descriptografia faz o inverso usando o expoente privado.

```python
def encrypt(self, message):
    # 1 - E
    # 0 - N
    encoded_message = [ord(char) for char in message]
    encrypted_message = [pow(ch, self._publicKey[1], self._publicKey[0]) for ch in encoded_message]

    return encrypted_message

1 usage  heitorpagotto
def decrypt(self, encrypted_message):
    # 1 - D
    # 0 - N
    encoded_message = [pow(ch, self._privateKey[1], self._privateKey[0]) for ch in encrypted_message]
    decrypted_message = "".join(chr(cha) for cha in encoded_message)

    return decrypted_message
```

b. Socket

Foi implementado a funcionalidade de hostear e de conectar em um host nas funções `run_socket_server` e `run_socket_client.`
Ambos os métodos começam criando um socket TCP, usando o método `socket.socket()`.
No servidor começamos a aceitar conexões e configuramos o socket em um IP e porta informados por parâmetro:

```python
2 usages  heitorpagotto +1
def run_socket_server(host, port):
    # Cria instancia da classe gerando uma chave privada e publica unica
    sv_rsa = RsaCryptography()
    sv_pub_key = sv_rsa.get_public_key()

    # Cria conexão socket com um host e porta
    sock = socket.socket()
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, value: 1)
    sock.bind((host, port))
    sock.listen(1)
    print(f"Player 1 ouvindo > {host}:{port}")

    conn, addr = sock.accept()

    print(f"Player 1 conectado > {addr}")
```

No cliente apenas conectamos no IP e porta informados por parametro:

```python
def run_socket_client(host, port):
    sleep(1)
    # Cria instancia da classe gerando uma chave privada e publica unica
    cl_rsa = RsaCryptography()
    cl_pub_key = cl_rsa.get_public_key()

    # Cria conexão socket com um host e porta
    sock = socket.socket()
    sock.connect((host, port))

    print(f"Player 2 conectado > {host}:{port}")
```

É feito a troca de chaves públicas para permitir a criptografia de ambos os lados e é feito o envio do evento de Start para o início do jogo ocorrer.

```python
if client_data["type"] == "KEY_EXCH":
    client_pub_key = [client_data["n"], client_data["e"]]

    # envia informação da chave publica para o server conectado
    conn.sendall(json.dumps(Message( type: "KEY_EXCH", e=sv_pub_key[1], n=sv_pub_key[0]).__dict__).encode())

    # cria instancia colocando a chave pública do servidor para poder criptografar dados
    client_rsa = RsaCryptography(client_pub_key[0], client_pub_key[1])

    # confirma se cliente recebeu chave
    client_key_received_raw = conn.recv(1024)
    client_key_received = json.loads(client_key_received_raw)

    if client_key_received['type'] == "KEY_ACK":
        game = TicTacToeGame()
        player_symbol, opponent_symbol = 'X', 'O'

        print("Você é o jogador", player_symbol)

        conn.sendall(json.dumps(Message("START").__dict__).encode())

        game.play(conn, player_symbol, opponent_symbol, sv_rsa, client_rsa)
```

c. Jogo

O jogo também foi abstraído para uma classe. Toda instancia da classe criada a propriedade `table` é zerada para ser uma matriz com 9 posições vazias.

```python
class TicTacToeGame:
    heitorpagotto
    def __init__(self):
        self.table = [
            [' ', ' ', ' '],
            [' ', ' ', ' '],
            [' ', ' ', ' ']
        ]
```

O método play roda um loop onde a matriz é impressa, e é chamado um input para o usuário escolher um número de 0 a 8 correspondente às posições no gráfico.
Após uma jogada do jogador atual é feito um envio de mensagem passando a posição para o servidor ou cliente, e quando a mensagem é recebida, a sincronização de posição é feita.

```
    playing_game = True
    current_player = 'X'

    while playing_game:
        self._print_table()
        print(f"Vez do jogador: {current_player}")

        input_message = "Jogador " + current_player + " digite um número de 0 a 8:"

        if current_player == player_symbol:
            sleep(1)
            grid_place = -1

            while grid_place < 0 or grid_place > 8:
                try:
                    grid_place = int(input(input_message))
                except ValueError:
                    input_message = 'Entrada inválida. Digite um numero de 0 a 8.'
                    print("Entrada invalida. Digite um numero de 0 a 8.")
                    continue

            if not self._make_move(grid_place, player_symbol):
                input_message = "Posicao já ocupada. Digite um numero de 0 a 8."
                continue

            encrypted_move = opponent_rsa.encrypt(str(grid_place))
            sock.sendall(json.dumps(Message( type: "MOVE", data=encrypted_move).__dict__).encode())
        else:
```

```
else:
    server_raw_data = sock.recv(4096)
    message = json.loads(server_raw_data.decode())
    if message['type'] == "MOVE":
        encrypted = message['data']
        position = int(player_rsa.decrypt(encrypted))
        self._make_move(position, opponent_symbol)

if self._verify_victory(current_player):
    self._print_table()
    print(f"Jogador {current_player} venceu!")
    sleep(1)
    sock.sendall(json.dumps(Message( type: "END", result=current_player).__dict__).encode())
    break

if self._verify_draw():
    self._print_table()
    print("Empate de jogo")
    sleep(1)
    sock.sendall(json.dumps(Message( type: "END", result="DRAW").__dict__).encode())
    break

current_player = 'O' if current_player == 'X' else 'X'
```