Segurança Computacional - 2023.2
Gustavo Barbosa de Almeida - 202037589
David Herbert de Souza Brito - 200057405
Isabelle Alex dos Santos Basílio Caldas - 170105636

Relatório de Implementação RSA-OAEP

O código apresentado é uma implementação do algoritmo RSA-OAEP (Optimal Asymmetric Encryption Padding) em Python, usando orientação a objetos. O RSA-OAEP é um esquema de criptografia assimétrica que combina o RSA para encriptação e assinatura digital com o OAEP para padding. Aqui está uma análise do código:

Estrutura do Código:

1. Classes:

- **ChaveRSA:** Responsável por gerar chaves RSA, verificar primalidade e realizar operações relacionadas a chaves.
- **FerramentasCripto:** Contém métodos úteis para manipulação de dados, como conversão de bytes para inteiros, geração de máscaras e operações bitwise.
- **Encriptar:** Herda de FerramentasCripto e adiciona métodos específicos para a encriptação RSA-OAEP.
- **Desencriptar:** Herda de FerramentasCripto e adiciona métodos específicos para a decriptação RSA-OAEP.
- **Receptor:** Combina ChaveRSA e Desencriptar, responsável por receber a mensagem encriptada, decriptá-la e testar a assinatura.
- **Servidor** Facilita a comunicação entre Remetente e Receptor, armazenando a mensagem encriptada e a chave pública do remetente.
- **Remetente:** Combina Encriptar e ChaveRSA, responsável por encriptar a mensagem, criar a assinatura e enviá-las ao servidor.

2. Funcionalidades:

- Geração de Chaves: O código permite a geração de novas chaves RSA e a escolha de utilizar chaves já existentes, facilitando o reuso.

- Encriptação e Decriptação: Implementa os passos necessários para realizar a encriptação e decriptação de mensagens usando o algoritmo RSA-OAEP.

- Assinatura e Verificação: A assinatura é criada usando SHA-3 e é verificada durante a decriptação.

3. Uso de Orientação a Objetos:

- O código usa conceitos de orientação a objetos, como herança, encapsulamento e polimorfismo.
- As classes são bem definidas e têm responsabilidades específicas, facilitando a compreensão e manutenção do código.

4. **Interatividade com o Usuário:**
- O código interage com o usuário, permitindo que ele escolha entre encriptar ou decriptar uma mensagem e gerar ou utilizar chaves existentes.

5. **Manipulação de Arquivos:**
- O código lê e escreve em arquivos para armazenar chaves e mensagens, garantindo persistência e reusabilidade.

Análise do código:

Esta seção fornece uma análise detalhada de algumas partes específicas do código.

Geração de Chaves - Classe `ChaveRSA`

A geração de chaves é crucial para a segurança do algoritmo RSA. O método `gerar_chave` da classe `ChaveRSA` realiza esse processo. Aqui estão algumas explicações:

```python
def gerar_chave(self, p, q, e):
    '''Cria chave pública(e, n) e a chave privada
    (d, n)
    input:
        p, q = números primos grandes, inteiros
        e = expoente da chave pública, inteiro
    output:
        chave pública, chave privada
    '''
    assert self.eh_primo(p, 7) and self.eh_primo(q, 7)
    assert p != q
    n = p * q
    phi = (p - 1) * (q - 1)
    if e is not None:
        assert self.mdc(phi, e) == 1
    else:
        while True:
            e = random.randrange(1, phi)
            if self.mdc(e, phi) == 1:
                break
    d = self.modulo_inverso(e, phi)
    return ((e, n), (d, n))
```

n é calculado como o produto dos números primos ***p*** e ***q***.

phi é a função totiente de Euler, essencial para o ***RSA***.

O método gera o expoente público e se não for fornecido.

Ele verifica a condição ***mdc(phi, e) == 1*** para garantir que e seja coprimo com ***phi***.

O expoente privado ***d*** é calculado usando o método ***modulo_inverso.***

Encriptação RSA-OAEP - Classe Encriptar

O método oaep_codificar implementa o esquema de padding OAEP. Aqui estão alguns esclarecimentos:

```python
def oaep_codificar(self, mensagem, k, rotulo=b''):
        '''
        Usa o algoritmo oaep para encriptar a mensagem "mensagem"

        input:
            mensagem = mensagem, bytearray
            k = tamanho da chave pública em bytes, inteiro
        output:
            mensagem encriptada, bytearray
        '''
        tam_mensagem = len(mensagem)
        hash_l = self.sha3(rotulo)
        ps = b'\x00' * (k - tam_mensagem - 2 * self.tam_hash - 2)
        db = hash_l + ps + b'\x01' + mensagem
        seed = os.urandom(self.tam_hash)
        db_mascara = self.mgf(seed, k - self.tam_hash - 1)
        db_mascarado = self.xor_bitwise(db, db_mascara)
        seed_mascara = self.mgf(db_mascarado, self.tam_hash)
        seed_mascarado = self.xor_bitwise(seed, seed_mascara)
        return b'\x00' + seed_mascarado + db_mascarado
```

O método implementa a operação de padding OAEP conforme descrito no algoritmo.

os.urandom(self.tam_hash) gera uma semente aleatória.

mgf é usado para gerar máscaras apropriadas.

A função ***xor_bitwise*** realiza operações de XOR entre bytes.

Decriptação RSA-OAEP - Classe `Desencriptar`

O método oaep_decodificar realiza a operação inversa de padding OAEP:

```python
def oaep_decodificar(self, c, k, rotulo=b''):
        '''EME-OAEP decoding
        Usa o algoritmo oaep para decriptar a mensagem "c"

         input:
            c = mensagem, bytearray
            k = tamanho da chave privada em bytes, inteiro
        output:
            mensagem decriptada, bytearray
        '''
        hash_l = self.sha3(rotulo)
        _, seed_mascarada, db_mascarado = c[:1], c[1:1 +
                                                   self.tam_hash], c[1 + self.tam_hash:]
        seed_mascara = self.mgf(db_mascarado, self.tam_hash)
        seed = self.xor_bitwise(seed_mascarada, seed_mascara)
        db_mascara = self.mgf(seed, k - self.tam_hash - 1)
        db = self.xor_bitwise(db_mascarado, db_mascara)
        _hash_l = db[:self.tam_hash]
        assert hash_l == _hash_l
        i = self.tam_hash
        while i < len(db):
            if db[i] == 0:
                i += 1
                continue
            elif db[i] == 1:
                i += 1
                break
            else:
                raise Exception()
        mensagem = db[i:]
        return mensagem
```

O método realiza a operação inversa de padding OAEP para obter a mensagem original.

Pontos de Melhoria:

1. **Comentários e Documentação:**
   - Embora as funções e métodos tenham nomes descritivos, alguns comentários adicionais poderiam ser úteis para explicar partes específicas do código.

2. **Tratamento de Exceções:**
   - Poderia haver um tratamento mais robusto de exceções, especialmente ao lidar com a leitura de arquivos e operações relacionadas ao sistema de arquivos.

3. **Melhoria na Interface do Usuário:**
   - A interface do usuário poderia ser aprimorada com mensagens mais claras e instruções adicionais.

4. **Testes Unitários:**
   - Seria benéfico incluir testes unitários para garantir que as funções e métodos estão produzindo os resultados esperados.

Conclusão:

O código apresenta uma implementação completa do algoritmo RSA-OAEP, seguindo boas práticas de programação e orientação a objetos. Ele oferece funcionalidades de encriptação, decriptação, assinatura digital e verificação. Com algumas melhorias, como comentários adicionais e tratamento de exceções mais abrangente, o código pode ser ainda mais robusto e compreensível.