Análise de Desempenho Criptográfico

Aluno: Lucas Dal Pra Brascher
Data: 29/04/2025

📜 Descrição da Atividade

Esta atividade consiste em analisar o desempenho dos algoritmos de criptografia simétrica (AES) e de chave pública (RSA), utilizando diferentes tamanhos de chave e avaliando o tempo de execução em cada cenário.

O texto criptografado foi:

> "RSA: algoritmo dos professores do MIT: Rivest, Shamir e Adleman."

🚀 Tecnologias Utilizadas

- Python 3.x
- Biblioteca `rsa`
- Biblioteca `pycryptodome` ou `pycryptodomex`
- IDE: PyCharm

📂 Como Executar

1. Clone o projeto ou copie o script.
2. Instale as dependências:

   ```
   pip install rsa pycryptodome
   ```

   ou, caso necessário:

```
pip install rsa pycryptodomex
```

3. Execute o script no terminal do PyCharm:
```
python script.py
```
4. Faça 5 execuções manuais de cada teste, anotando o tempo de geração de chave e criptografia.
5. Tire prints de cada execução para o relatório.

⚙️ Configuração da Máquina de Testes

Item	Especificação
Sistema Operacional	macOS Sonoma (provável 14.x, confirme no menu Apple > Sobre este Mac)
Processador (CPU)	Apple M4 Pro
Frequência do CPU	~3.5 GHz (estimado, Apple Silicon não publica exato, mas é referência comum)
Núcleos da CPU	12 núcleos (8 de performance + 4 de eficiência, típico do M4 Pro)
Memória RAM	24 GB unificada
Armazenamento (SSD)	512 GB

📊 Resultados

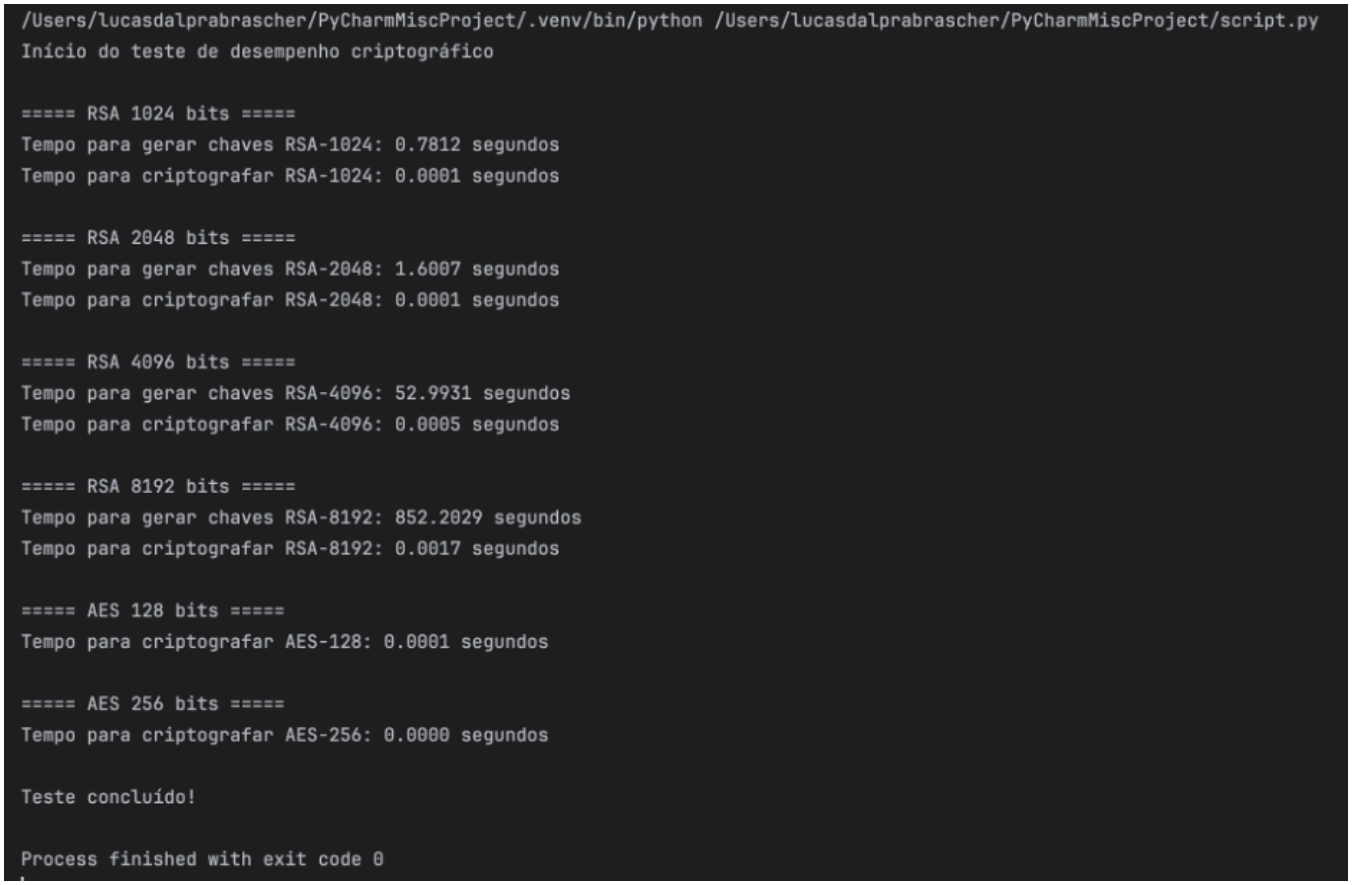
Algoritmo	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5	Média exata (s)
RSA-1024 (Geração)	0.0194 s	0.7812 s	1.0534 s	0.2909 s	0.2909 s	0.48716
RSA-1024 (Criptografia)	0.0001 s	0.0001 s	0.0001 s	0.0001 s	0.0000 s	0.00008
RSA-2048 (Geração)	2.7070 s	1.6007 s	0.6598 s	0.7931 s	0.8904 s	1.33020
RSA-2048 (Criptografia)	0.0001 s	0.0001 s	0.0002 s	0.0001 s	0.0001 s	0.00012
RSA-4096 (Geração)	54.8150 s	52.9931 s	150.8610 s	164.1260 s	111.9067 s	106.94036
RSA-4096 (Criptografia)	0.0005 s	0.0005 s	0.0005 s	0.0005 s	0.0005 s	0.00050
RSA-8192 (Geração)	1298.8105 s	852.2029 s	209.5055 s	771.4655 s	259.0520 s	678.20728
RSA-8192 (Criptografia)	0.0018 s	0.0017 s	0.0018 s	0.0018 s	0.0018 s	0.00178
AES-128 (Criptografia)	0.0001 s	0.0001 s	0.0000 s	0.0000 s	0.0000 s	0.00004
AES-256 (Criptografia)	0.0000 s	0.0000 s	0.0000 s	0.0000 s	0.0000 s	0.00000

📸 Prints de Execução

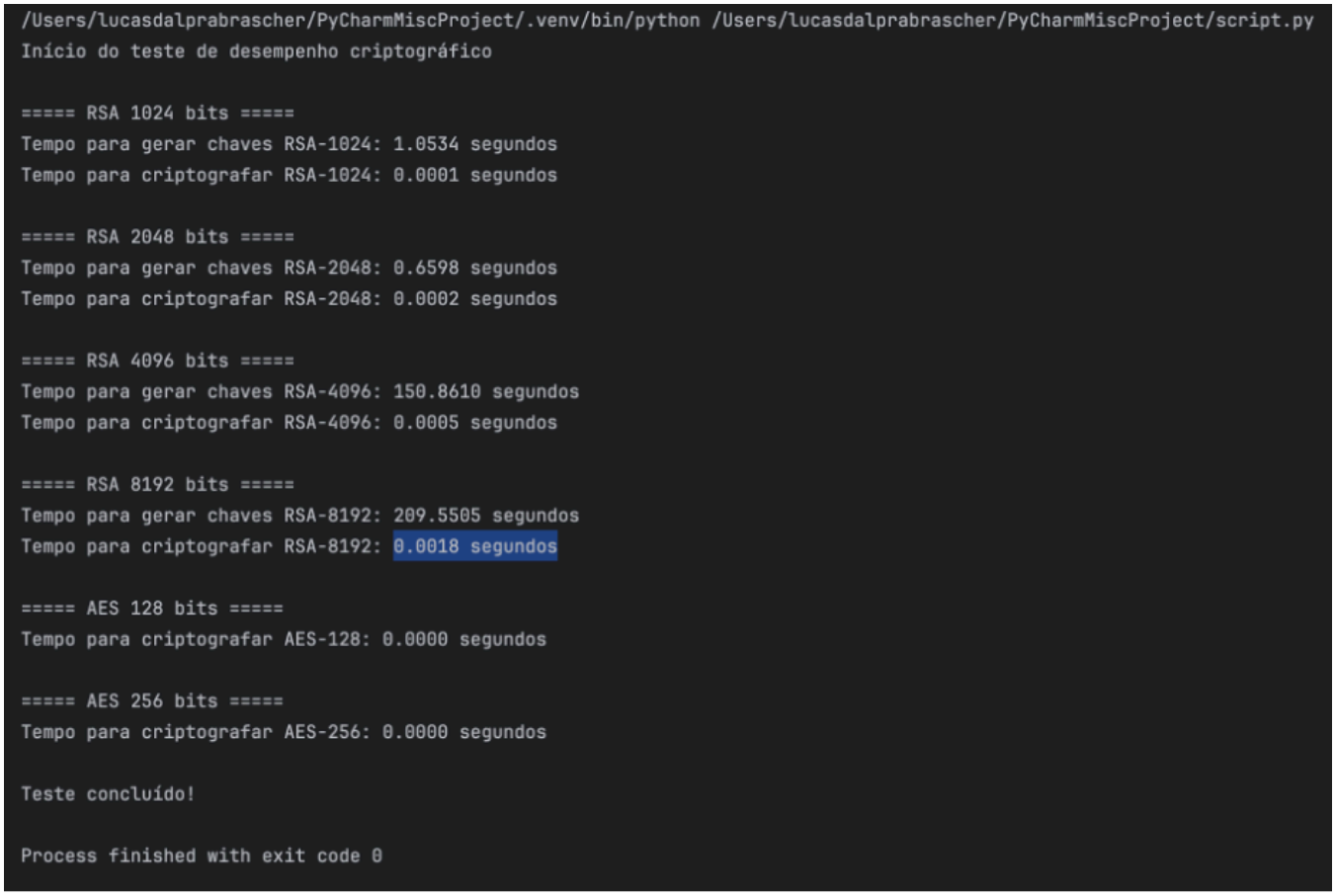
Execução 1:

```
/Users/lucasdalprabrascher/PyCharmMiscProject/.venv/bin/python /Users/lucasdalprabrascher/PyCharmMiscProject/script.py
Início do teste de desempenho criptográfico

===== RSA 1024 bits =====
Tempo para gerar chaves RSA-1024: 0.0194 segundos
Tempo para criptografar RSA-1024: 0.0000 segundos

===== RSA 2048 bits =====
Tempo para gerar chaves RSA-2048: 2.7070 segundos
Tempo para criptografar RSA-2048: 0.0001 segundos

===== RSA 4096 bits =====
Tempo para gerar chaves RSA-4096: 54.8150 segundos
Tempo para criptografar RSA-4096: 0.0005 segundos

===== RSA 8192 bits =====
Tempo para gerar chaves RSA-8192: 1298.8105 segundos
Tempo para criptografar RSA-8192: 0.0018 segundos

===== AES 128 bits =====
Tempo para criptografar AES-128: 0.0001 segundos

===== AES 256 bits =====
Tempo para criptografar AES-256: 0.0000 segundos

Teste concluído!

Process finished with exit code 0
```

Execução 2:


/Users/lucasdalprabrascher/PyCharmMiscProject/.venv/bin/python /Users/lucasdalprabrascher/PyCharmMiscProject/script.py
Início do teste de desempenho criptográfico
===== RSA 1024 bits =====
Tempo para gerar chaves RSA-1024: 0.7812 segundos
Tempo para criptografar RSA-1024: 0.0001 segundos
===== RSA 2048 bits =====
Tempo para gerar chaves RSA-2048: 1.6007 segundos
Tempo para criptografar RSA-2048: 0.0001 segundos
===== RSA 4096 bits =====
Tempo para gerar chaves RSA-4096: 52.9931 segundos
Tempo para criptografar RSA-4096: 0.0005 segundos
===== RSA 8192 bits =====
Tempo para gerar chaves RSA-8192: 852.2029 segundos
Tempo para criptografar RSA-8192: 0.0017 segundos
===== AES 128 bits =====
Tempo para criptografar AES-128: 0.0001 segundos
===== AES 256 bits =====
Tempo para criptografar AES-256: 0.0000 segundos
Teste concluído!
Process finished with exit code 0

Execução 3:


/Users/lucasdalprabrascher/PyCharmMiscProject/.venv/bin/python /Users/lucasdalprabrascher/PyCharmMiscProject/script.py
Início do teste de desempenho criptográfico
===== RSA 1024 bits =====
Tempo para gerar chaves RSA-1024: 1.0534 segundos
Tempo para criptografar RSA-1024: 0.0001 segundos
===== RSA 2048 bits =====
Tempo para gerar chaves RSA-2048: 0.6598 segundos
Tempo para criptografar RSA-2048: 0.0002 segundos
===== RSA 4096 bits =====
Tempo para gerar chaves RSA-4096: 150.8610 segundos
Tempo para criptografar RSA-4096: 0.0005 segundos
===== RSA 8192 bits =====
Tempo para gerar chaves RSA-8192: 209.5505 segundos
Tempo para criptografar RSA-8192: 0.0018 segundos
===== AES 128 bits =====
Tempo para criptografar AES-128: 0.0000 segundos
===== AES 256 bits =====
Tempo para criptografar AES-256: 0.0000 segundos
Teste concluído!
Process finished with exit code 0

Execução 4:

```
/Users/lucasdalprabrascher/PyCharmMiscProject/.venv/bin/python /Users/lucasdalprabrascher/PyCharmMiscProject/script.py
Início do teste de desempenho criptográfico

===== RSA 1024 bits =====
Tempo para gerar chaves RSA-1024: 0.2909 segundos
Tempo para criptografar RSA-1024: 0.0001 segundos

===== RSA 2048 bits =====
Tempo para gerar chaves RSA-2048: 0.7931 segundos
Tempo para criptografar RSA-2048: 0.0001 segundos

===== RSA 4096 bits =====
Tempo para gerar chaves RSA-4096: 164.1260 segundos
Tempo para criptografar RSA-4096: 0.0005 segundos

===== RSA 8192 bits =====
Tempo para gerar chaves RSA-8192: 771.4655 segundos
Tempo para criptografar RSA-8192: 0.0018 segundos

===== AES 128 bits =====
Tempo para criptografar AES-128: 0.0001 segundos

===== AES 256 bits =====
Tempo para criptografar AES-256: 0.0000 segundos

Teste concluído!

Process finished with exit code 0
```

Execução 5:

```
/Users/lucasdalprabrascher/PyCharmMiscProject/.venv/bin/python /Users/lucasdalprabrascher/PyCharmMiscProject/script.py
Início do teste de desempenho criptográfico

===== RSA 1024 bits =====
Tempo para gerar chaves RSA-1024: 0.1175 segundos
Tempo para criptografar RSA-1024: 0.0000 segundos

===== RSA 2048 bits =====
Tempo para gerar chaves RSA-2048: 0.8904 segundos
Tempo para criptografar RSA-2048: 0.0001 segundos

===== RSA 4096 bits =====
Tempo para gerar chaves RSA-4096: 11.9067 segundos
Tempo para criptografar RSA-4096: 0.0005 segundos

===== RSA 8192 bits =====
Tempo para gerar chaves RSA-8192: 259.0520 segundos
Tempo para criptografar RSA-8192: 0.0018 segundos

===== AES 128 bits =====
Tempo para criptografar AES-128: 0.0001 segundos

===== AES 256 bits =====
Tempo para criptografar AES-256: 0.0000 segundos

Teste concluído!

Process finished with exit code 0
```

📜 Código Fonte

O script utilizado encontra-se no arquivo `script.py`.

📌 Importações

```python
import time
import rsa
import base64
from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes
```

- `time`: para medir os tempos de execução das operações.
- `rsa`: biblioteca usada para gerar chaves e cifrar com RSA.
- `base64`: usada para codificações, embora não tenha sido utilizada no código final.

- `Cryptodome.Cipher.AES` : biblioteca para criptografia simétrica (AES).
- `get_random_bytes` : usada para gerar chaves aleatórias para o AES.

📌 Texto base para os testes

```
texto = "RSA: algoritmo dos professores do MIT: Rivest, Shamir e Adleman".encode('utf-8')
```

- Define o texto que será criptografado por todos os algoritmos.
- O `.encode('utf-8')` transforma a string em bytes, pois os algoritmos exigem esse formato.

🔐 Função RSA: `rsa_encrypt_test(bits)`

```
(public_key, private_key) = rsa.newkeys(bits)
```

- Gera um par de chaves (pública e privada) com o número de bits fornecido (ex: 1024, 2048...).

```
encrypted_text = rsa.encrypt(texto, public_key)
```

- Criptografa o texto com a chave pública RSA gerada.

```
time.time()
```

- Mede o tempo antes e depois da operação para calcular a duração da geração da chave e da criptografia.

🔐 Função AES: `aes_encrypt_test(bits)`

```
key = get_random_bytes(bits // 8)
```

- Gera uma chave AES aleatória (ex: 128 ou 256 bits), convertendo para bytes (dividido por 8).

```
cipher = AES.new(key, AES.MODE_EAX)
```

- Cria um objeto de cifra no modo `EAX` (modo autenticado e seguro).

```
ciphertext, tag = cipher.encrypt_and_digest(texto)
```

- Criptografa o texto e gera uma `tag` de verificação de integridade.

▶ Execução principal

```
if __name__ == "__main__":
```

- Ponto de entrada do script quando ele é executado diretamente.

```
rsa_encrypt_test(1024)
rsa_encrypt_test(2048)
rsa_encrypt_test(4096)
rsa_encrypt_test(8192)
```

- Executa os testes com criptografia RSA em quatro tamanhos de chave.

```
aes_encrypt_test(128)
aes_encrypt_test(256)
```

- Executa os testes com criptografia AES com chaves de 128 e 256 bits.

✅ Resultado

- O script imprime no console o tempo gasto para gerar chaves RSA e criptografar com RSA e AES.
- Os dados devem ser registrados manualmente em prints e planilhas.

🔗 Referências

- Documentação da biblioteca rsa
- Documentação da biblioteca pycryptodome
- StackOverflow para resolução de dúvidas específicas