Uma imagem com texto

Descrição gerada automaticamente

**Performance Benchmarking of Cryptographic Mechanisms**

**Relatório 1**

João Sousa e Rui Santos

CC2009: Segurança e Privacidade

Prof.º Manuel Correia

Prof.º João Vilela

Prof.º Henrique Faria

Março de 2024

Índice

[Implementação 2](#_Toc161853826)

[Descrição código para alínea B 2](#_Toc161853827)

[Imports 2](#_Toc161853828)

[Função de encriptação 2](#_Toc161853829)

[Função de desencriptação 3](#_Toc161853830)

[Funções auxiliares 3](#_Toc161853831)

[Descrição código para alínea C 5](#_Toc161853832)

[Imports 5](#_Toc161853833)

[Criação de chave publica e chave privada 6](#_Toc161853834)

[Função de encriptação 6](#_Toc161853835)

[Função de desencriptação 7](#_Toc161853836)

[Funções auxiliares 7](#_Toc161853837)

[Descrição código para alínea D 9](#_Toc161853838)

[Imports 9](#_Toc161853839)

[Geração de hash 9](#_Toc161853840)

[Funções auxiliares 10](#_Toc161853841)

[Explain how you generated/obtained the results 11](#_Toc161853842)

[Plots showing 11](#_Toc161853843)

[Bibliografia 15](#_Toc161853844)

# Implementação

De seguida temos a descrição dos componentes principais para cada porção de código utilizada para as alíneas B, C e D do projeto.É importante referir que em todo o projeto utilizamos a biblioteca de criptografia desenvolvida por a organização PyCA.

## Descrição código para alínea B

Para melhor descrever o código decidimos dividir em diferentes secções e explicar cada uma delas com uma imagem do código e uma breve descrição.

### Imports

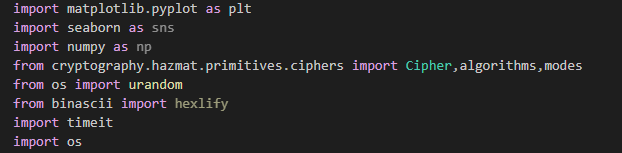


Figura 1- imports alínea B

### Função de encriptação

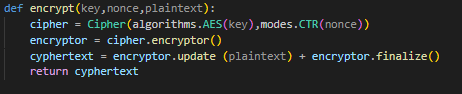


Figura 2 - função de encriptação

Para a encriptação criamos uma função que recebe os argumentos:

* Key : palavra chave secreta (terá sempre 256 bits);
* Nonce : utilizado pelo modo CTR;
* Plaintext: texto a encriptar.

No final a função devolve o texto cifrado.

### Função de desencriptação

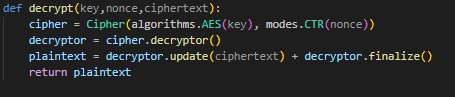


Figura 3 - função de desencriptação

Para a desencriptação criamos uma função que recebe os argumentos:

* Key : palavra chave secreta (terá sempre 256 bits);
* Nonce : utilizado pelo modo CTR;
* ciphertext: texto a desencriptar.

No final a função devolve o texto original.

### Funções auxiliares

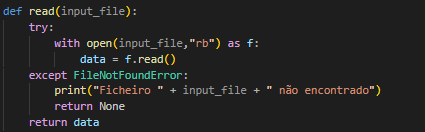


Figura 4 - função de leitura de ficheiro

Função utilizado para leitura de dados presentes num ficheiro passado como argumento. Em caso de sucesso, retorna os conteúdos de input\_file sobre a forma de sequência de bytes. Em caso de insucesso retorna None.

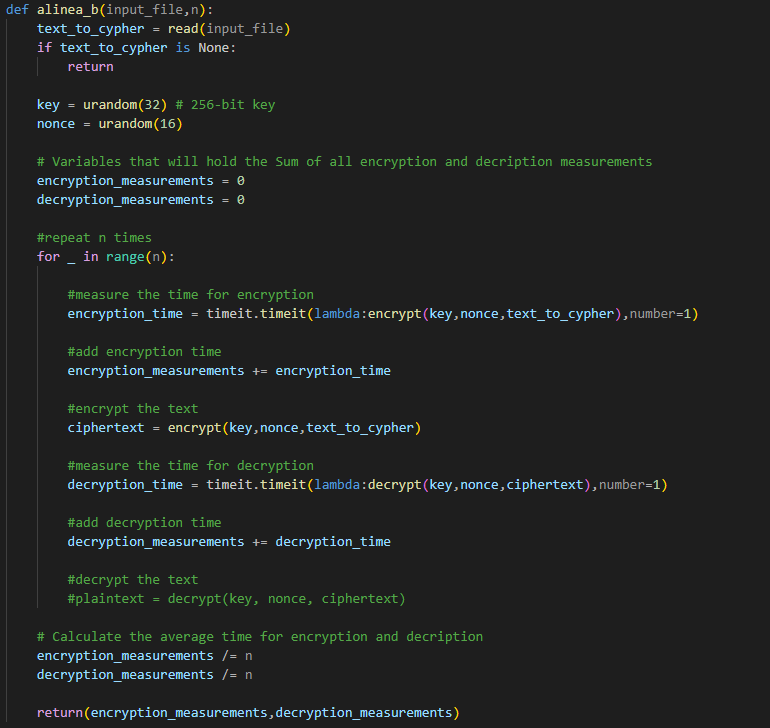


Figura 5 - função que implementa a parte “principal” da alínea B do projeto

Esta função tem como objetivo:

* Receber ficheiro de input, que vai conter o texto a ser cifrado;
* Fazer a encriptação e decriptação desse texto n vezes;
* Em cada iteração calcular o tempo que demorou a encriptar e a desencriptar (individualmente);
* Adicionar os resultados de cada iteração as respetivas variáveis auxiliares;
* Por fim , após as n iterações , fazer a média da encriptação e decriptação e retornar estes valores.

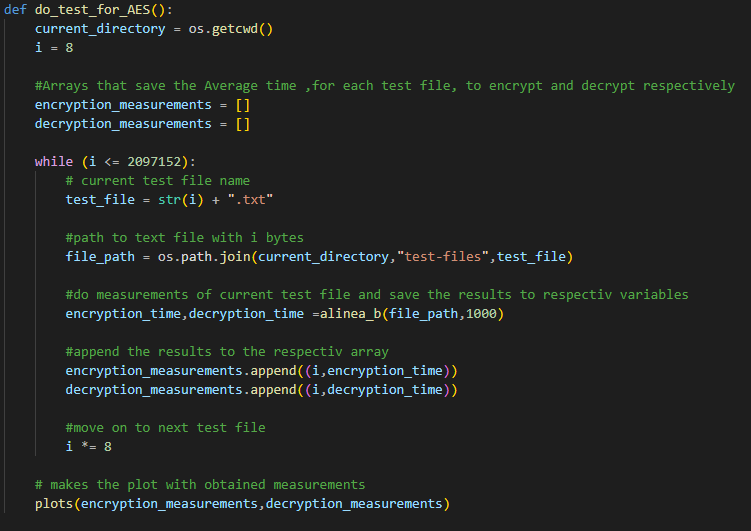


Figura 7 - função que realiza todos os testes definidos para a alínea B

Função que realizei os testes em todos os casos de teste, mencionados na alínea A do projeto para o algoritmo AES. É importante referir que nos realizamos 1000 repetições para garantir resultados estatisticamente significantes. No final são feitos os gráficos.

## Descrição código para alínea C

Mais uma vez, para melhor descrever o código decidimos dividir em diferentes secções e explicar cada uma delas com uma imagem do código e uma breve descrição.

### Imports

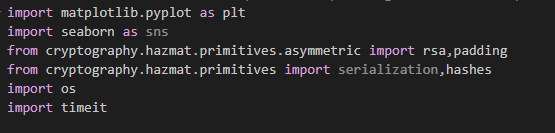


Figura 8- imports alínea C

### Criação de chave publica e chave privada

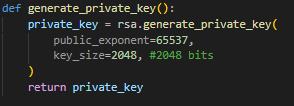


Figura 9 - função que gera chave privada e variáveis globais que representam chave privada e chave publica

Para a criação da chave privada, definimos uma função que queria uma chave privada com o tamanho da chave igual a 2048 bits e retornamos essa chave privada. É importante referir que utilizamos variáveis globais que guardam a chave privada e chave pública, e não tivemos em conta este tempo de geração da chave privada para os tempo medidos.

### Função de encriptação

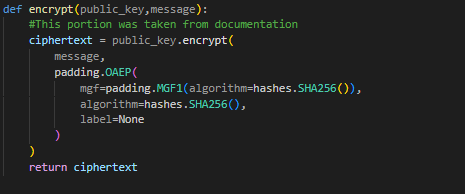


Figura 10 - função de encriptação

Para a encriptação criamos uma função que recebe os argumentos:

* Public\_key : palavra chave publica ,usada para a encriptação (obtida através da chave privada );
* message: mensagem que vamos encriptar.

Neste caso seguimos a encriptação com os parâmetros sugeridos pelos criadores da biblioteca.

### Função de desencriptação

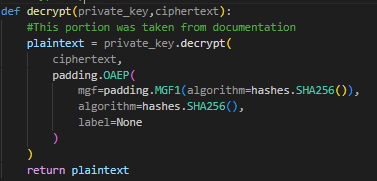


Figura 11 - função de desencriptação

Para a desencriptação criamos uma função que recebe os argumentos:

* Private\_key : palavra chave privada ,usada para a desencriptação;
* Ciphertext : texto cifrado que vamos desencriptar.

Neste caso seguimos também os parâmetros sugeridos pelos criadores da biblioteca.

### Funções auxiliares

Decidimos não acrescentar as funções read porque é exatamente igual a definidas na alínea B.

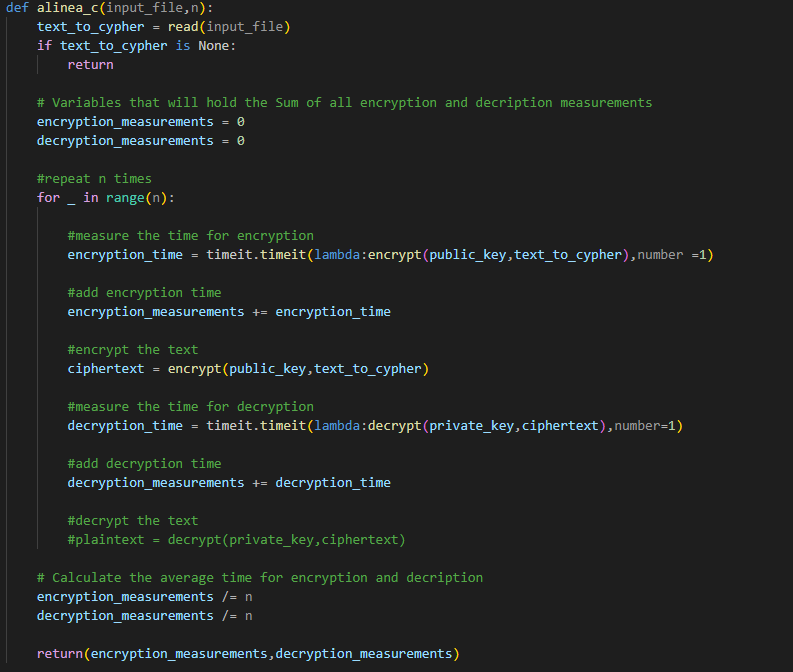


Figura 12 - função que implementa a parte “principal” da alínea C do projeto

Esta função tem como objetivo:

* Receber ficheiro de input, que vai conter o texto a ser cifrado;
* Fazer a encriptação e decriptação desse texto n vezes;
* Em cada iteração calcular o tempo que demorou a encriptar e a desencriptar (individualmente);
* Adicionar os resultados de cada iteração nas variáveis auxiliares respetivas;
* Por fim, após as n iterações , fazer a média da encriptação e decriptação e retornar estes valores.

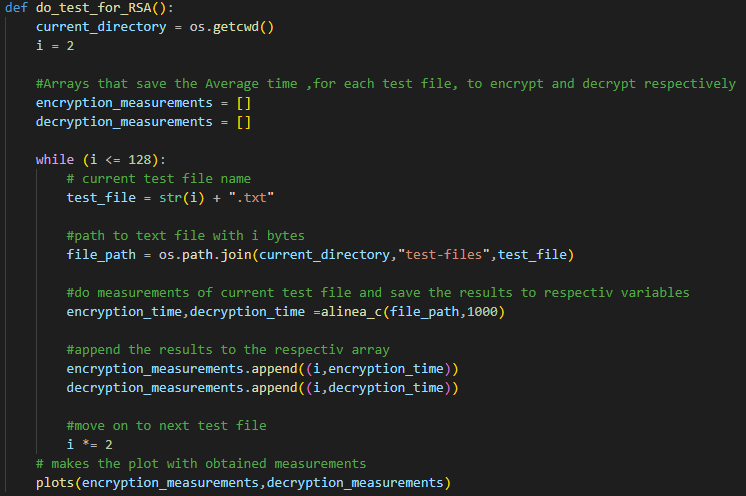


Figura 13 - função que realiza todos os testes definidos para a alínea C

Função que realizei os testes em todos os casos de teste mencionados na alínea A do projeto para o algoritmo RSA. É importante referir que nos realizamos 1000 repetições para garantir resultados estatisticamente significantes. No final são feitos os gráficos.

## Descrição código para alínea D

Tal como fizemos anteriormente, vamos descrever o código em diferentes secções e explicar cada uma delas com uma imagem do código e uma breve descrição.

### Imports

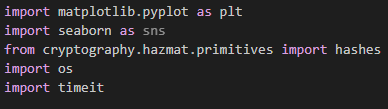


Figura 14 - imports alínea D

### Geração de hash

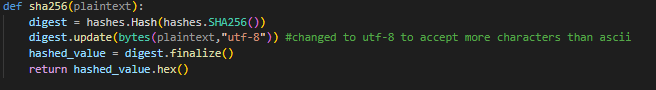


Figura 15 - função que gera hash SHA-256

Recebe como argumento o texto em que vamos aplicar a função de hashing.No final retorna a resultante da aplicação da função hash em Hexadecimal.

### Funções auxiliares

Decidimos não acrescentar as funções read porque é exatamente igual a definidas na alínea B.

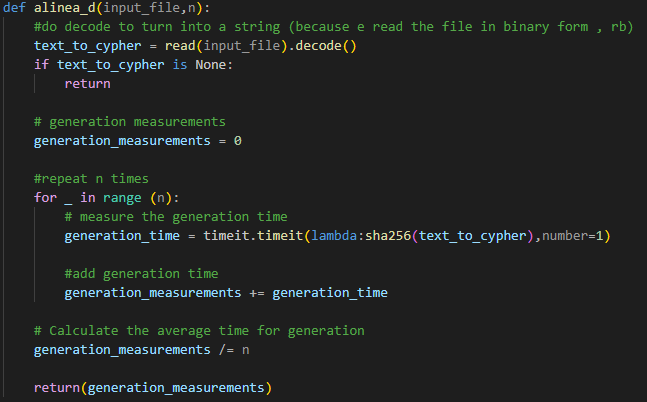


Figura 16 - função que implementa a parte “principal” da alínea D do projeto

Esta função tem como objetivo:

* Receber ficheiro de input, que vai conter o texto que vamos aplicar a função de hash;
* Aplicar a função de hash a esse texto n vezes;
* Em cada iteração calcular o tempo que demorou a gerar a hash ;
* Adiciona o resultado de cada iteração numa variável auxiliar;
* Por fim, após as n iterações, fazer a média geração e retorna este valor.

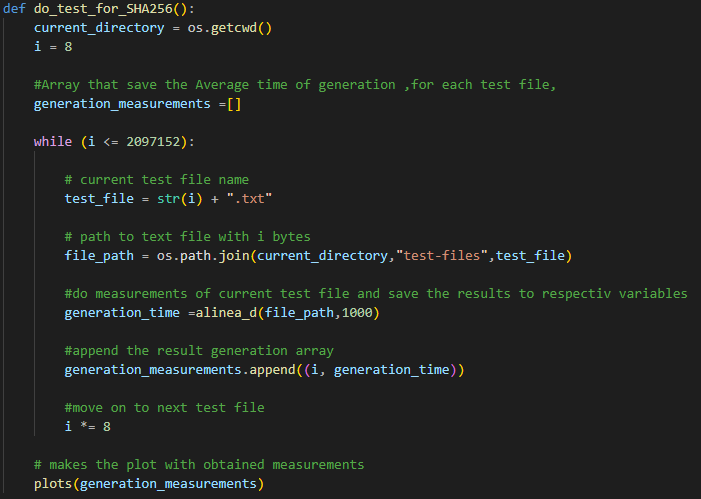


Figura 17- função que realiza todos os testes definidos para a alínea D

Função que realizei os testes em todos os casos de teste mencionados na alínea A do projeto para o algoritmo SHA-256. É importante referir que nos realizamos 1000 repetições para garantir resultados estatisticamente significantes. No final são feitos os gráficos.

# Explain how you generated/obtained the results

POR FAZER

# Plots showing

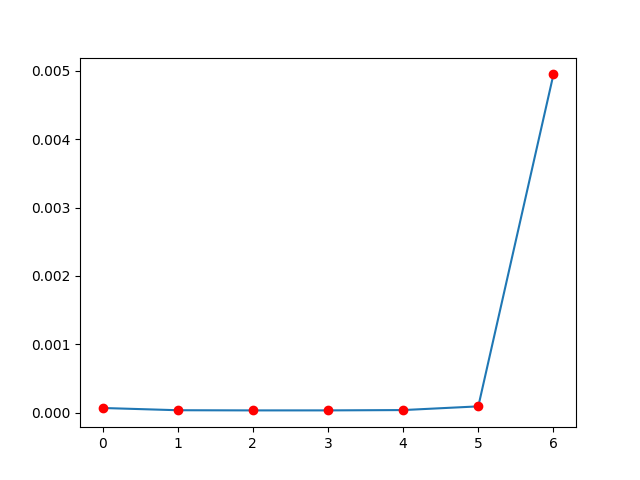


Figure 1 - Gráfico da média dos tempos de encriptação para todos os ficheiros pedidos

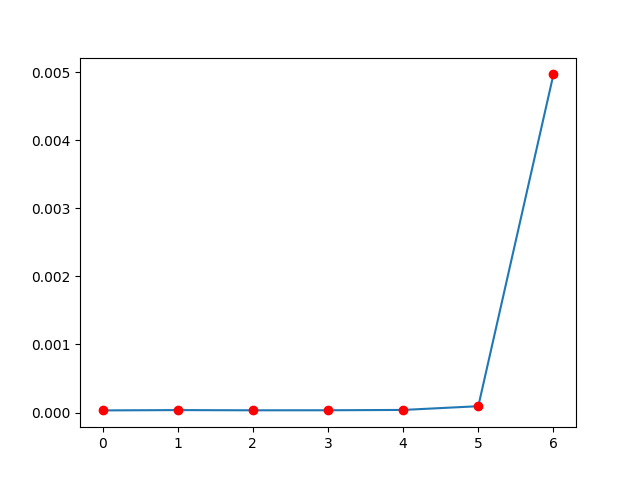
AES\_ENCRYPTION\_TIMES AES\_DECRYPTION\_TIMES

Figure 2 - Gráfico da média dos tempos de decriptação para todos os ficheiros pedidos

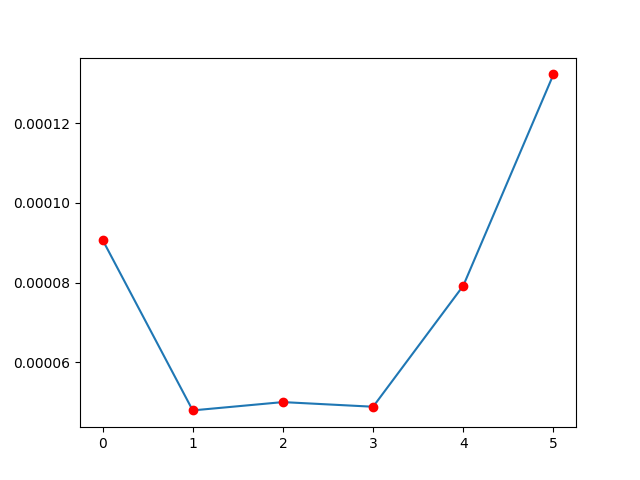


Figure 3 - Gráfico da média dos tempos de encriptação para todos os ficheiros pedidos exceto 2097152 bytes(por motivos de escala)

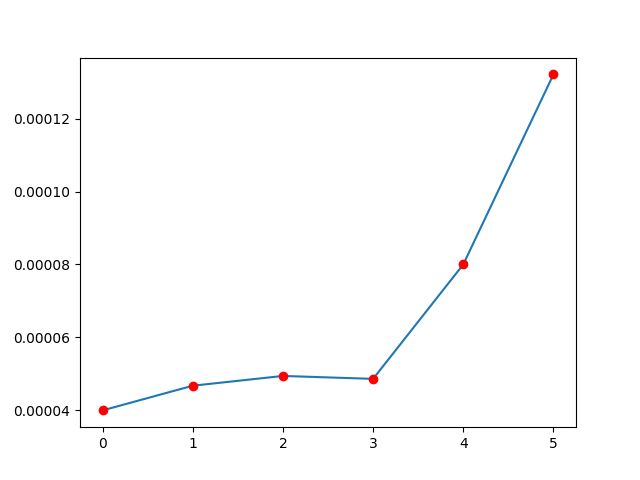


Figure 4 - Gráfico da média dos tempos de decriptação para todos os ficheiros pedidos exceto 2097152 bytes(por motivos de escala)

|  |  |  |
| --- | --- | --- |
| Valor de X no gráfico | Tamanho em bytes correspondente | Como espectável os tempos de encriptação desencriptação são muito semelhantes sendo um  algoritmo bastante rápido até 262144 bytes com  um tempo próximo de 50 µs existindo um aumento significativo de tempo de execução no ficheiro de 2097152 para 5000 µs |
| 0 | 8 |
| 1 | 64 |
| 2 | 512 |
| 3 | 4096 |
| 4 | 32786 |
| 5 | 262144 |
| 6 | 2097152 |

RSA\_ENCRYPTION\_TIMES RSA\_DECRYPTION\_TIMES

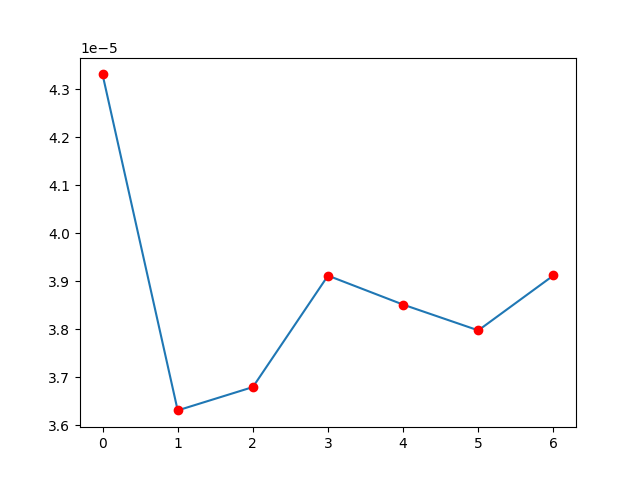
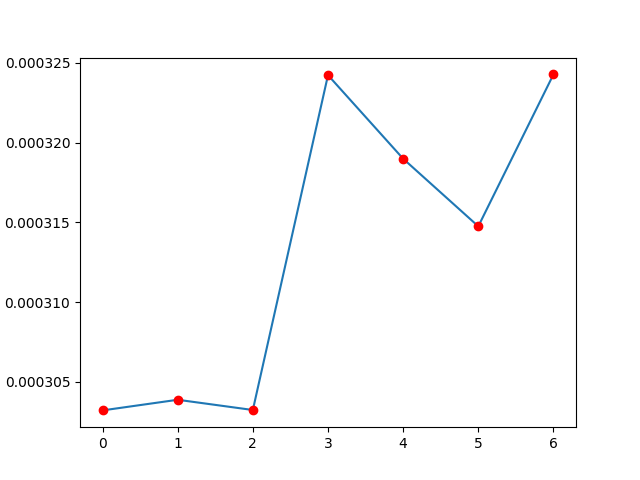


Figure 5 - Gráfico da média dos tempos de decriptação para todos os ficheiros pedidos

Figure 6 - Gráfico da média dos tempos de encriptação para todos os ficheiros pedidos

|  |  |  |
| --- | --- | --- |
| Valor de X no gráfico | Tamanho em bytes correspondente | Como espectável os tempos de encriptação desencriptação não dependem do tamanho do ficheiro sendo o tempo de encriptação na ordem dos40 µs já o tempo de desencriptação é da casa dos 300 µs quase 100 vezes mais lento |
| 0 | 2 |
| 1 | 4 |
| 2 | 8 |
| 3 | 16 |
| 4 | 32 |
| 5 | 64 |
| 6 | 128 |

SHA\_GENERATION\_TIMES

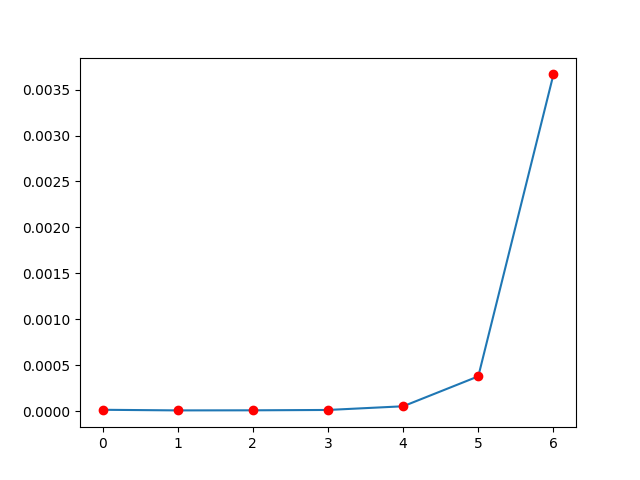
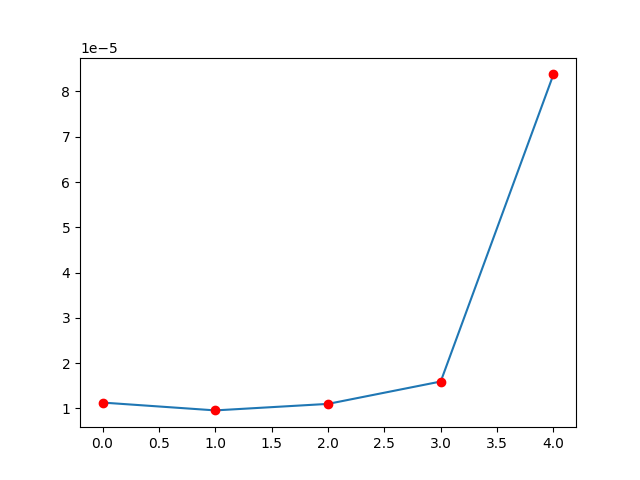


Figure 7 - - Gráfico da média dos tempos de geração para todos os ficheiros pedidos exceto,262144 e 2097152 bytes(por motivos de escala)

Figure 8 - - Gráfico da média dos tempos de geração para todos os ficheiros pedidos

|  |  |  |
| --- | --- | --- |
| Valor de X no gráfico | Tamanho em bytes correspondente | Os tempos de geração de SHA para ficheiros até  4096 bytes é de 1 µs sendo que existe um aumento para 8 µs para 32786 bytes, para 262144 bytes 500 µs e 20987152 bytes 3500 µs |
| 0 | 8 |
| 1 | 64 |
| 2 | 512 |
| 3 | 4096 |
| 4 | 32786 |
| 5 | 262144 |
| 6 | 2097152 |

# Bibliografia

* <https://cryptography.io/en/latest/hazmat/primitives/>
* <https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/>
* <https://cryptography.io/en/latest/hazmat/primitives/symmetric-encryption/>
* <https://ioflood.com/blog/timeit-python/>
* <https://docs.python.org/3/library/timeit.html>
* <https://stackoverflow.com/questions/18458734/how-do-i-plot-list-of-tuples>