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](#_Toc161343698)

[Descrição código para alínea B 2](#_Toc161343699)

[Imports 2](#_Toc161343700)

[Função de encriptação 2](#_Toc161343701)

[Função de desencriptação 3](#_Toc161343702)

[Funções auxiliares 3](#_Toc161343703)

[Descrição código para alínea C 5](#_Toc161343704)

[Imports 5](#_Toc161343705)

[Criação de chave publica e chave privada 6](#_Toc161343706)

[Função de encriptação 6](#_Toc161343707)

[Função de desencriptação 6](#_Toc161343708)

[Funções auxiliares 7](#_Toc161343709)

[Descrição código para alínea D 9](#_Toc161343710)

[Imports 9](#_Toc161343711)

[Geração de hash 9](#_Toc161343712)

[Funções auxiliares 9](#_Toc161343713)

[Explain how you generated/obtained the results 10](#_Toc161343714)

[Plots showing 10](#_Toc161343715)

[Bibliografia 11](#_Toc161343716)

# 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

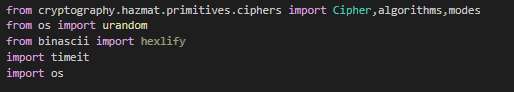


Figura 1- imports alínea B

### Função de encriptação

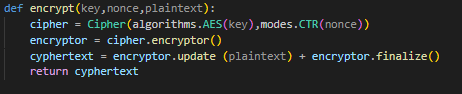


Figura 2 - função de encriptação

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

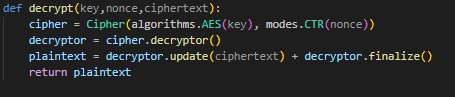


Figura 3 - função de desencriptação

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

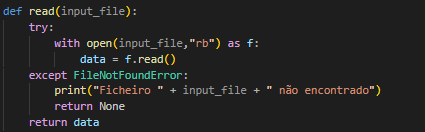


Figura 4 - função de leitura de ficheiro

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.

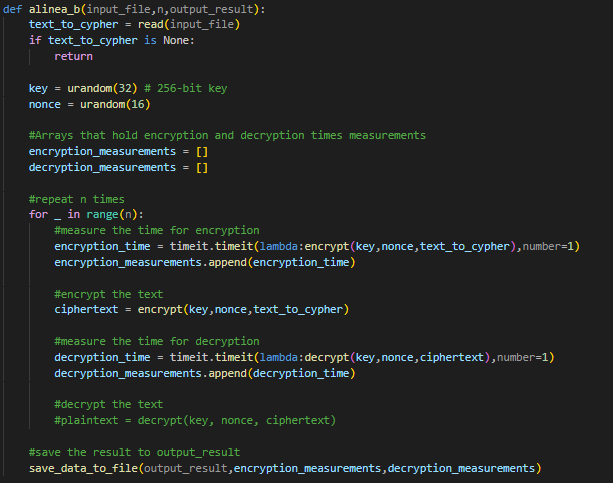


Figura 5 - função que implementa a parte “principal” da alínea B do projeto

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);
* Guardar os resultados de cada iteração em arrays auxiliares ;
* Por fim , após as n iterações , guardar esses arrays num ficheiro dado por output\_result.

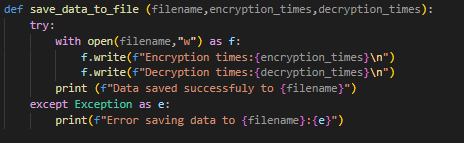


Figura 6 - função que guarda resultados das medidas num ficheiro

Guarda os resultados calculados pela função descrita anteriormente, num ficheiro passado como argumento. Também imprime mensagem em caso de sucesso ou insucesso ao guardar esses dados.

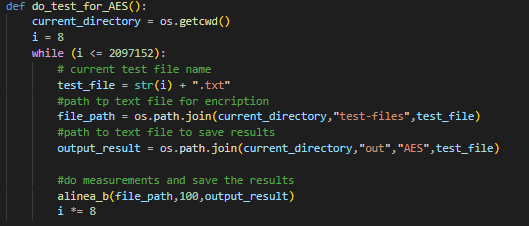


Figura 7 - função que realiza todos os testes definidos para a alínea B

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 100 repetições para garantir resultados estatisticamente significantes.

## 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

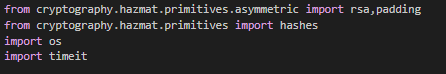


Figura 8- imports alínea C

### Criação de chave publica e chave privada

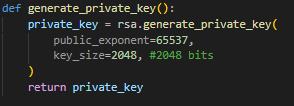


Figura 9 - função que gera chave privada e variáveis globais que representam chave privada e chave publica

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 publica , e não tivemos em conta este tempo de geração da chave privada para os tempo medidos.

### Função de encriptação

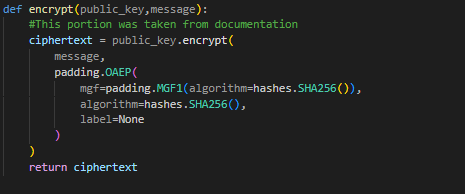


Figura 10 - função de encriptação

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

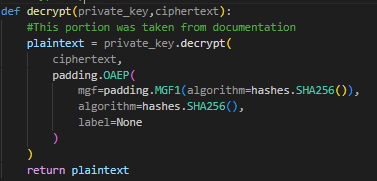


Figura 11 - função de desencriptação

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 e save\_data\_to\_file porque são exatamente iguais as definidas na alínea B.

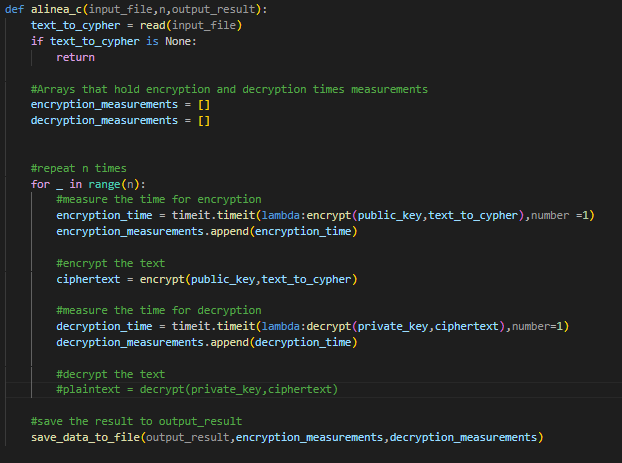


Figura 12 - função que implementa a parte “principal” da alínea C do projeto

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);
* Guardar os resultados de cada iteração em arrays auxiliares ;
* Por fim , após as n iterações , guardar esses arrays num ficheiro dado por output\_result.

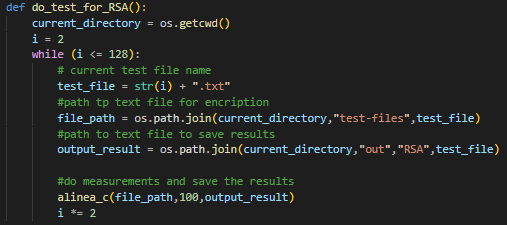


Figura 13 - função que realiza todos os testes definidos para a alínea C

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 100 repetições para garantir resultados estatisticamente significantes.

## 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

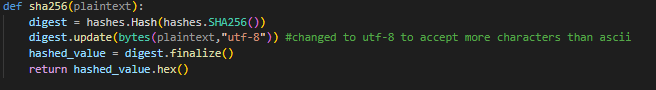


Figura 15 - função que gera hash SHA-256

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 e save\_data\_to\_file são quase iguais as definidas na alínea B.

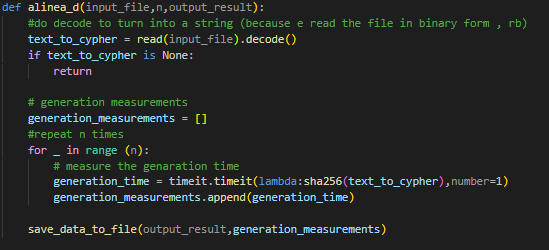


Figura 16 - função que implementa a parte “principal” da alínea D do projeto

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 ;
* Guardar os resultados de cada iteração num array auxiliare ;
* Por fim, após as n iterações , guardar esse array num ficheiro dado por output\_result.

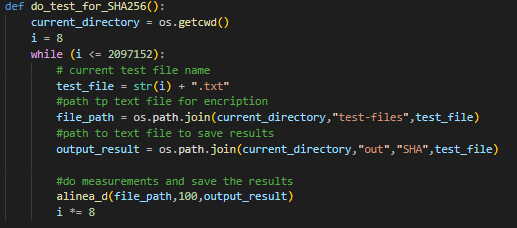


Figura 17- função que realiza todos os testes definidos para a alínea D

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 100 repetições para garantir resultados estatisticamente significantes

# Explain how you generated/obtained the results

POR FAZER

# Plots showing

POR FAZER

# 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