Uma imagem com texto, captura de ecrã, Gráficos, design gráfico

Os conteúdos gerados por IA poderão estar incorretos.

-

ÍNDICE

* **Introdução**
* **Explicação do Código**  
    - Instalação das Dependências  
    - Importação das Bibliotecas  
    - Geração de Dados Aleatórios para Ficheiros  
    - Função para o Desenho dos Gráficos  
    - AES: Encriptação e Desencriptação  
    - RSA: Encriptação e Desencriptação  
    - SHA-256: Geração de Hash  
    - Análise de Dados  
    - Relevância Estatística da Medição do Tempo
* **Análise Estatística dos Resultados**  
    - AES: Encriptação e Desencriptação  
    - RSA: Encriptação e Decriptação  
    - SHA-256: Geração de Hash  
    - Comparação entre os três  
    - Comparação entre Ambientes de Trabalho e Configurações
* **Relevância Estatística da Medição do Tempo**

 - AES: Encriptação e Desencriptação  
  - RSA: Encriptação e Dessencriptação  
  - SHA-256: Geração de Hash

* **Conclusão**
* **Referências**

Introdução

Neste trabalho, vamos medir o tempo que o AES (Advanced Encryption Standard), o RSA (Rivest–Shamir–Adleman) e o SHA-256 (Secure Hash Algorithm 256) demoram a processar ficheiros de tamanhos diferentes, utilizando uma implementação em Python para os mecanismos de encriptação, desencriptação e hashing.

A criptografia é uma das principais ferramentas da segurança da informação moderna, e é

essencial para garantir a confidencialidade, integridade e autenticidade dos dados nos

sistemas digitais. Com o crescimento exponencial da quantidade de dados transferidos e

armazenados diariamente, tornou-se essencial não só garantir a segurança, mas também

avaliar o desempenho dos mecanismos criptográficos utilizados.

De seguida iremos à descrição individual de cada um dos mecanismos:

-AES (encriptação simétrica) – A encriptação simétrica é um método criptográfico em que a mesma chave é utilizada tanto para encriptar como para desencriptar os dados. Reconhecido pela sua velocidade e eficiência em ambientes de comunicação seguros.

-RSA (encriptação assimétrica) – A encriptação assimétrica utiliza um par de chaves: uma chave pública para encriptação e uma chave privada para desencriptação. A chave pública pode ser partilhada livremente, enquanto a chave privada deve ser mantida segura. Destaca-se pela sua segurança na troca de chaves, porém tem um elevado custo computacional.

-SHA-256 (função de hash) – O hashing é uma função unidirecional que recebe um input (ou “mensagem”) e devolve uma cadeia de bytes de tamanho fixo. É essencial para garantir a integridade dos dados e é um dos pilares da tecnologia Blockchain.

O objetivo deste trabalho é obter uma compreensão prática sobre o comportamento destes algoritmos.

O trabalho foi desenvolvido em linguagem Python (3.13.2), em formato de jupyter notebook utilizando o editor VS Code.

Explicação do código

- Instalação das Dependências

Primeiramente instalamos as dependências necessárias para a realização do trabalho:

- Cryptography: Usada para a realização de operações de encriptação, desencriptação e geração de chaves.

- MatPlotLib: Utilizada para a criação de gráficos e visualização de dados.

- Numpy: Fornecimento de ferramentas para cálculos numéricos eficientes e manipulação de arrays.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Os conteúdos gerados por IA poderão estar incorretos.- Importação das Bibliotecas

A biblioteca cryptography foi uma das principais, fornecendo as ferramentas necessárias para a encriptação e desencriptação dos algoritmos AES e RSA, através dos módulos ‘ciphers’ e ‘asymmetrics’. Já hashlib foi usada para gerar os hashes para o algoritmo SHA-256, de forma simples e eficiente. Para medir tempos de execução, foram utilizadas as bibliotecas timeit e time. Statistics permitiu calcular as métricas essenciais na análise dos resultados. A matplotlib.pyplot foi fundamental na criação dos gráficos que representam visualmente os desempenhos dos algoritmos, e a numpy, incluindo ‘trapz’, serviu para operações matemáticas e análise numérica. Já as bibliotecas os e random foram usadas para simular dados e controlar o ambiente de execução. Todas estas bibliotecas, em conjunto, tornaram possível a implementação robusta e a análise detalhada dos testes realizados.

Uma imagem com texto, captura de ecrã, software, ecrã

Os conteúdos gerados por IA poderão estar incorretos.- Geração de Dados Aleatórios para Ficheiros

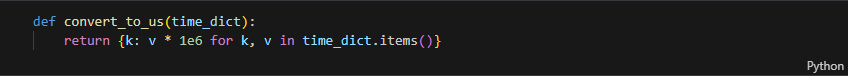
Utilizamos a biblioteca os com a função os.random(size) para a criação de sequências de bytes, para a simulação de ficheiros realistas, dos seguintes tamanhos:

Para AES (em bytes): 8, 64, 512, 4096, 32768, 262144, 2097152;

Para SHA (em bytes): 8, 64, 512, 4096, 32768, 262144, 2097152;

Para RSA (em bytes): 2, 4, 8, 16, 32, 64, 128.

Em seguida esses dados foram organizados em dicionários, onde cada chave representa o tamanho do ficheiro, e o valor associado corresponde ao conteúdo aleatório gerado.

- Função para o Desenho dos Gráficos

Esta função tem como finalidade converter os tempos de execução de segundos para microssegundos, que vão ser utilizados nos desenhos de gráficos futuramente.

Uma imagem com captura de ecrã, texto, software

Os conteúdos gerados por IA poderão estar incorretos.- AES: Encriptação e Desencriptação

Neste passo avançamos para a implementação do algoritmo de encriptação simétrica AES

(Advanced Encryption Standard).

O AES foi configurado no modo CTR (Counter Mode), que permite encriptação por blocos sem necessidade de padding, o que é ideal para medições de desempenho justas e diretas.

A chave foi gerada com ‘os.urandom(32)`, produzindo uma chave de 256 bits (32 bytes).

Foi utilizado o módulo ‘Cipher’ da biblioteca ‘cryptography.hazmat’, combinado com o modo ‘modes.CTR(iv)’ e o backend padrão.

A operação de encriptação e desencriptação foi feita com ‘encryptor.update(data) +

encryptor.finalize()’ e respetivamente para o decryptor, garantindo que todos os blocos

sejam processados corretamente. A escolha pelo modo CTR deve-se ao seu desempenho eficiente e ao facto de não necessitar de padding.

Esta decisão permite comparar diretamente o desempenho com os algoritmos RSA e SHA-256, que também não usam padding. A utilização de ‘os.urandom` para gerar dados e IVs garante uma aleatoriedade segura e replicável.

A medição de tempos foi realizada com timeit.Timer, repetindo cada operação 100 vezes para obter médias representativas e comparáveis.

- RSA: Encriptação e Desencriptação

Uma imagem com texto, captura de ecrã, software

Os conteúdos gerados por IA poderão estar incorretos.

Seguimos com a implementação do algoritmo de encriptação assimétrica RSA (Rivest–Shamir–Adleman).

Ao contrário da encriptação simétrica, o RSA baseia-se em operações matemáticas envolvendo números primos e exponenciação modular, permitindo o uso de chaves públicas e privadas distintas para encriptar e desencriptar respetivamente.

Foi gerado um par de chaves RSA com 2048 bits, utilizando a função rsa.generate\_private\_key, que fornece automaticamente a chave pública e os parâmetros necessários (n, e, d).

A implementação segue uma abordagem de RSA “raw”, em que os dados são convertidos diretamente de bytes para inteiros, e a operação de encriptação é realizada com pow(m, e, n), enquanto a desencriptação é feita com pow(ciphertext, d, n).

A ausência de padding nesta versão simplificada permite medir o desempenho puro do algoritmo, sem interferência de estruturas adicionais como OAEP ou PKCS#1.

A medição de tempos foi realizada com timeit.Timer, repetindo cada operação 100 vezes para obter médias representativas e comparáveis.

- SHA: Geração de Hash

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Finalizamos com a implementação do algoritmo de hashing SHA-256 (Secure Hash Algorithm 256 bits).

O SHA-256 é um algoritmo determinístico, rápido e irreversível, adequado para aplicações criptográficas modernas.

A função de hashing foi implementada com a biblioteca padrão do Python ‘hashlib’ que fornece suporte essencial ao algoritmo SHA-256.

O processo consistiu em criar um objeto de hash com ‘hashlib.sha256()`, atualizar com os dados do ficheiro simulado e obter o resultado com o método ‘digest()`.

Como se trata de um algoritmo puramente computacional, não foram utilizados parâmetros externos como chaves ou IV’s.

A medição de tempos foi realizada com timeit.Timer, repetindo cada operação 100 vezes para obter médias representativas e comparáveis.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.- Análise de Dados

Inicialmente criamos um gráfico com o propósito de comparar o tempo de encriptação de dados entre os algoritmos AES e RSA e o tempo de hashing do SHA-256, usando ficheiros de tamanhos diferentes, utilizando ‘matplotlib’.

O gráfico facilita a comparação entre os algoritmos, mostrando como o tempo de processamento varia com o tamanho dos dados.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

De seguida criamos um gráfico com o propósito de comparar o tempo de desencriptação dos dados entre os algoritmos AES e RSA, usando ficheiros de tamanhos diferentes, utilizando ‘matplotlib’.

O gráfico facilita a comparação entre os algoritmos, mostrando como o tempo de processamento varia com o tamanho dos dados.

- Relevância Estatística da Medição do Tempo

Uma imagem com texto, captura de ecrã, software

Os conteúdos gerados por IA poderão estar incorretos.

Para garantir a relevância estatística na medição dos tempos cada operação foi repetida 100 vezes por tamanho de ficheiro, tanto com dados fixos como aleatórios. Esta repetição permitiu recolher um número suficiente de amostras para calcular métricas fiáveis, reduzindo o impacto de variações momentâneas no sistema.

A medição foi feita com time.perf\_counter() e convertida para microssegundos para maior precisão.

A mesma lógica foi utilizada para os restantes algoritmos testados, alterando apenas as funções de encriptação ou desencriptação e os tamanhos de ficheiro.

Esta consistência na estrutura do código garantiu uma comparação equilibrada entre os diferentes algoritmos e a produção de resultados estatisticamente significativos.

Análise Estatística dos Resultados

Para esta parte utilizamos dois ambientes de testes e configurações diferentes, para verificar se os tempos de encriptação, desencriptação e hashing alteravam, de modo a analisar o impacto do hardware no desempenho dos algoritmos.

Os dois computadores utilizados nos testes têm o mesmo sistema operativo (Windows 11) e estão equipados com 16 GB de RAM.

O primeiro computador tem um Intel Core i7-12700H (12.ª geração), enquanto o segundo usa um i7-13620H (13.ª geração). Todas as imagens à esquerda serão representativas do 1º computador.

- AES: Encriptação e Desencriptação

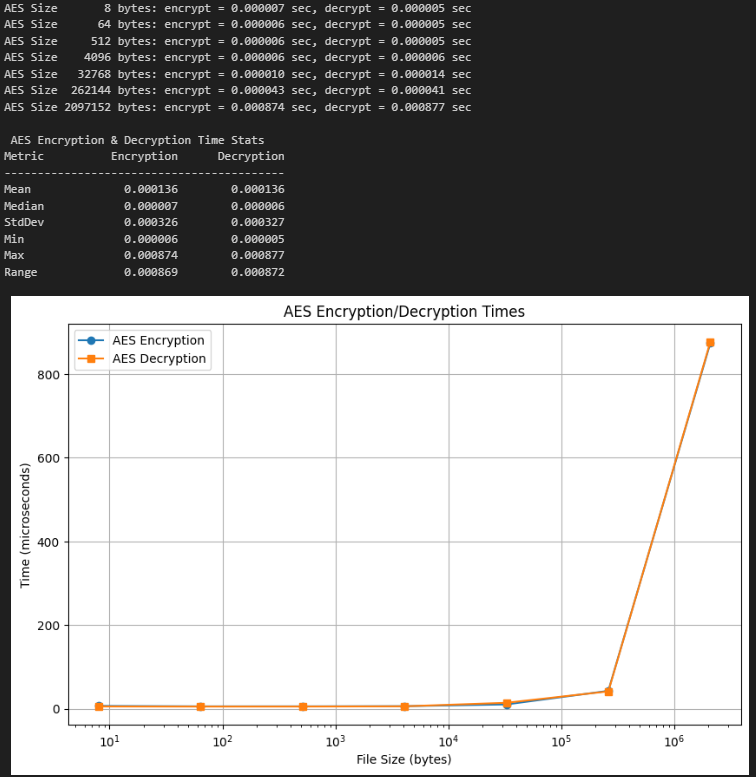
Uma imagem com texto, captura de ecrã

Os conteúdos gerados por IA poderão estar incorretos.

Esta parte do código serve para a ilustração do desempenho da encriptação e desencriptação do algoritmo AES, através de valores estatísticos relevantes e de um gráfico, usando ‘matplotlib’.

Uma imagem com texto, captura de ecrã, Gráfico, número

Os conteúdos gerados por IA poderão estar incorretos.



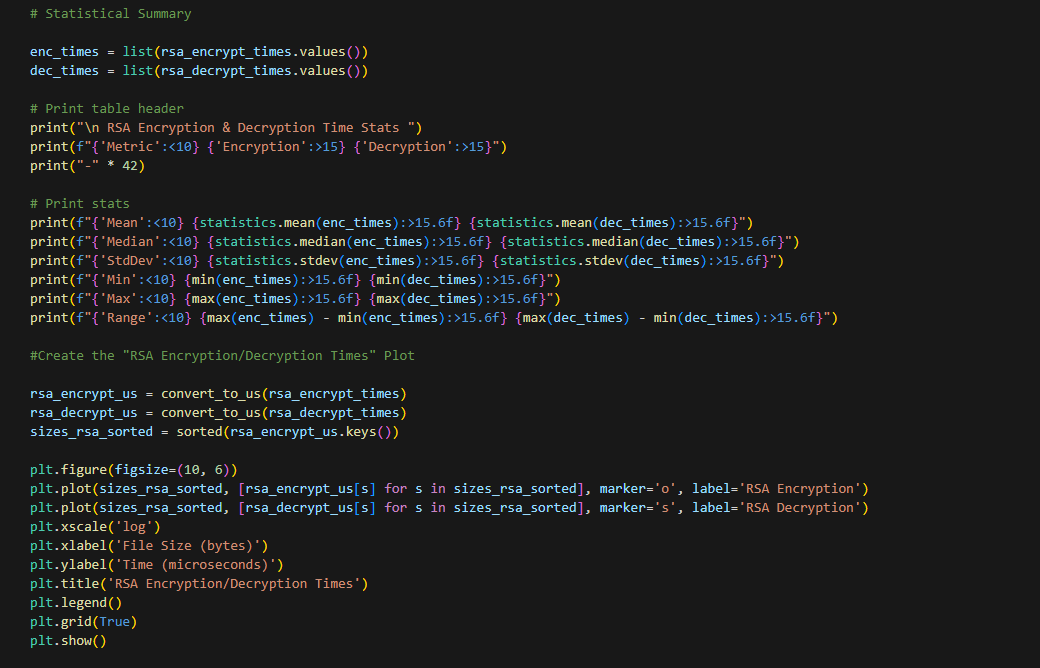
Com base nos dados e nos gráficos apresentados, observa-se que o algoritmo AES apresenta uma performance bastante eficiente, tanto na encriptação como na desencriptação, com tempos de execução muito baixos para ficheiros pequenos e um crescimento gradual à medida que o tamanho dos dados aumenta também.

A curva visível no gráfico mantém-se relativamente estável nas primeiras medidas, começando apenas a subir significativamente com ficheiros de grande dimensão, o que demonstra boa escalabilidade.

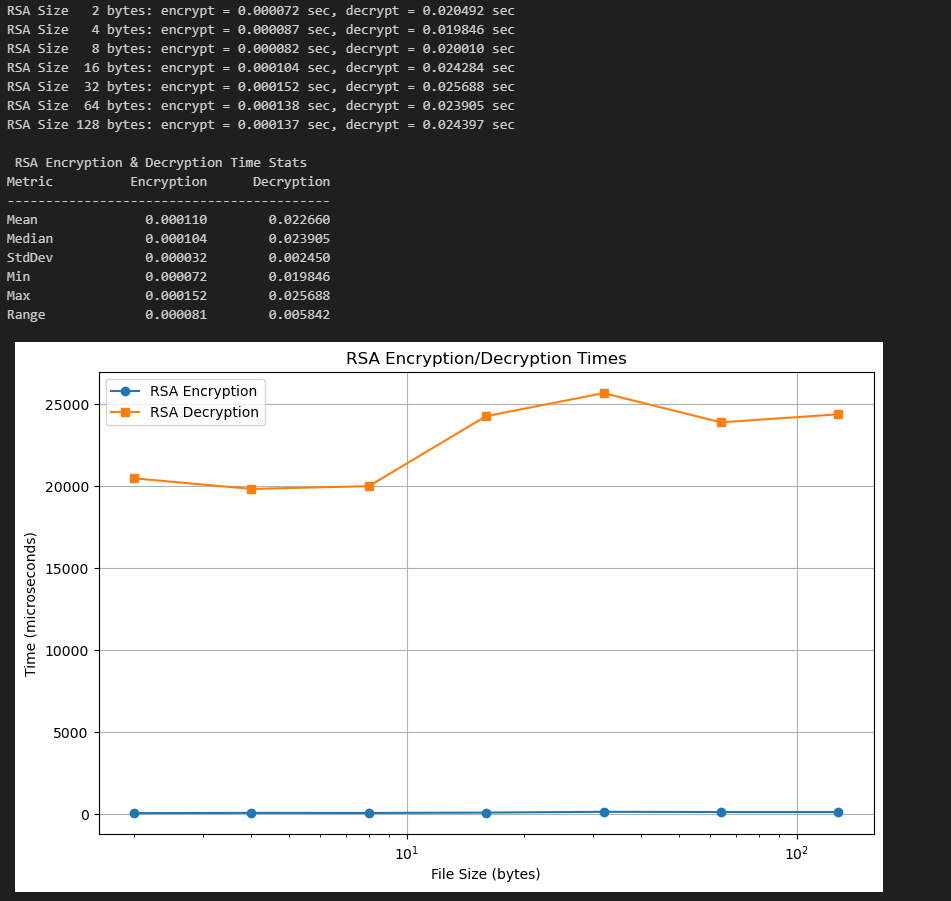
Outro ponto a destacar é a semelhança entre os tempos de encriptação e desencriptação, que acompanham a mesma tendência em todos os tamanhos, reforçando a simetria e equilíbrio de desempenho do algoritmo.

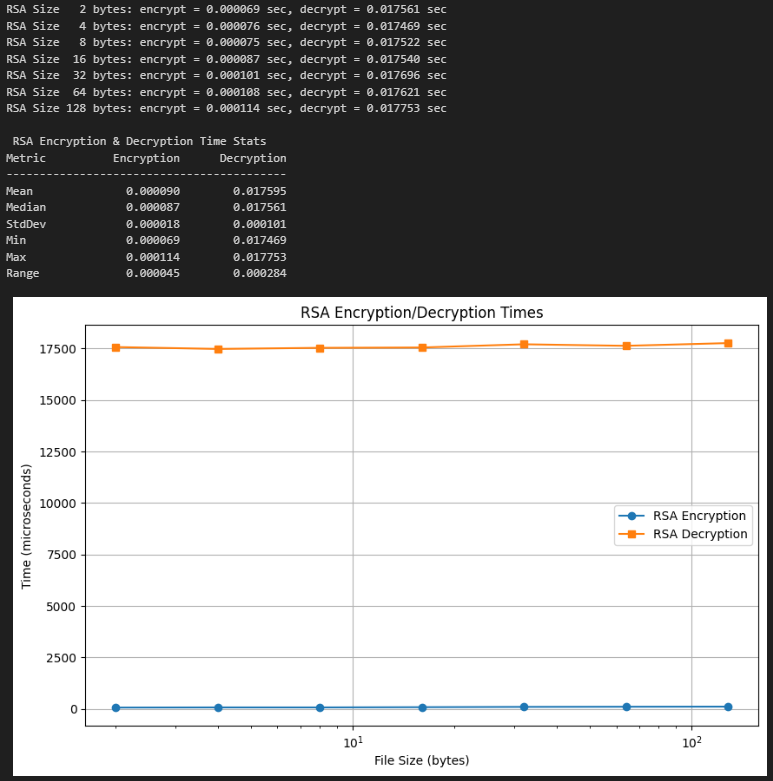
Esta estabilidade torna o AES particularmente adequado para aplicações que exigem rapidez e consistência, mesmo em ambientes com diferentes volumes de dados.

- RSA: Encriptação e Desencriptação



Esta parte do código serve para a ilustração do desempenho da encriptação e desencriptação do algoritmo RSA, através de valores estatísticos relevantes e de um gráfico, usando ‘matplotlib’.





Através da análise dos gráficos e dos tempos apresentados, é possível verificar que o algoritmo RSA demonstra um comportamento assimétrico entre encriptação e desencriptação, sendo a desencriptação significativamente muito mais demorada.

Os tempos de encriptação mantêm-se relativamente constantes com o aumento do tamanho dos ficheiros, indicando eficiência e estabilidade no processo de codificação.

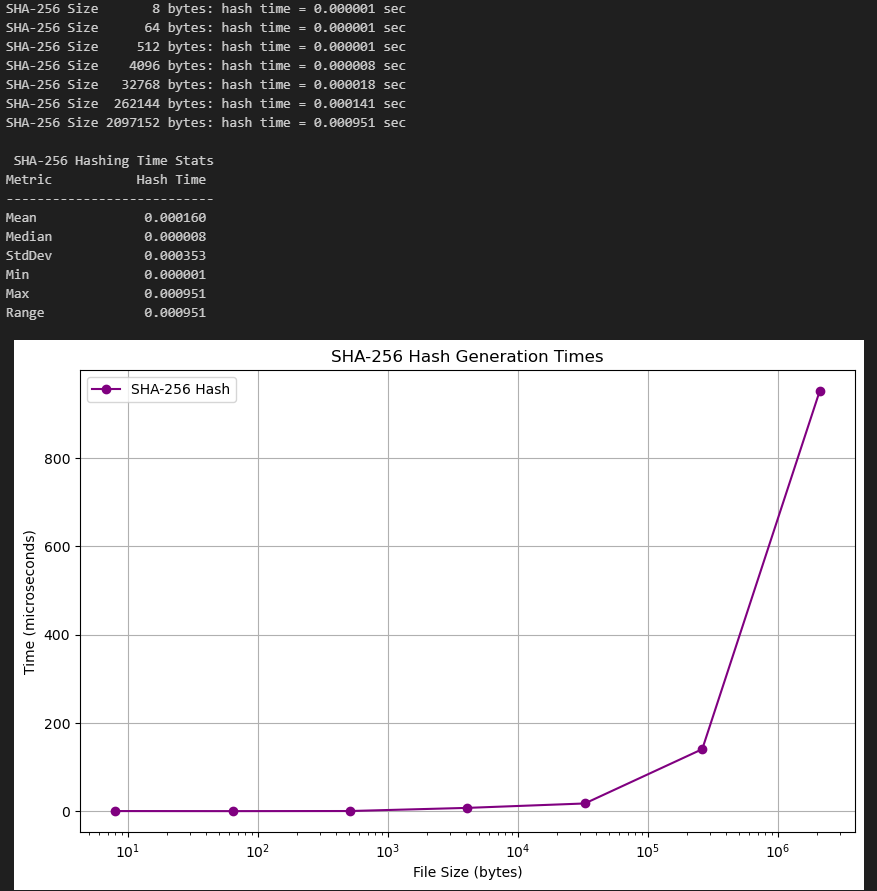
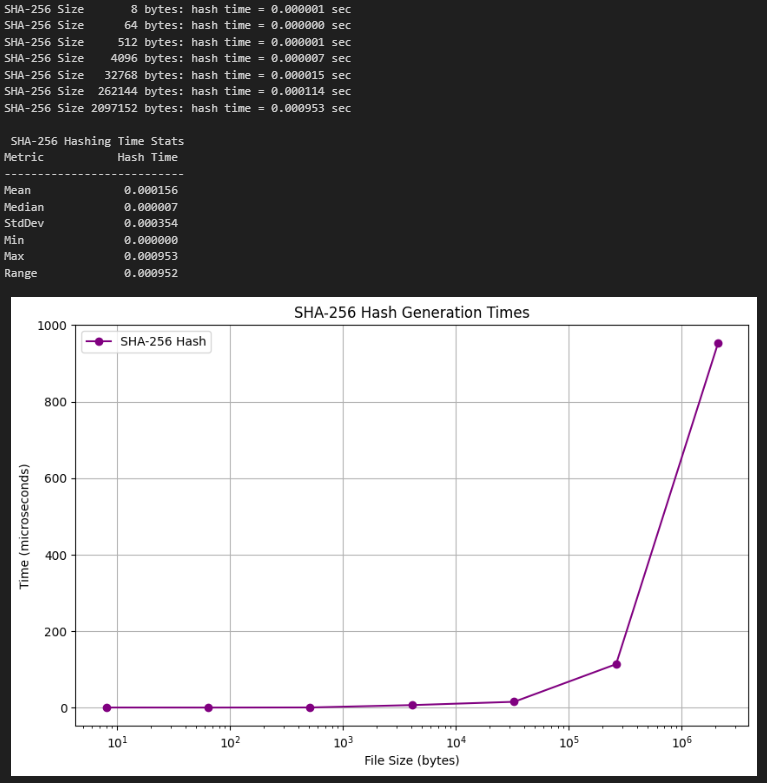
Por outro lado, a desencriptação exibe tempos muito mais elevados, com variações ligeiras entre tamanhos, o que está de acordo com a natureza computacionalmente intensiva da operação de desencriptação no RSA.

Esta diferença confirma que o RSA, sendo um algoritmo assimétrico, é mais adequado para operações pontuais de segurança, como troca de chaves, do que para encriptação de grandes volumes de dados.

Uma imagem com texto, captura de ecrã

Os conteúdos gerados por IA poderão estar incorretos.- SHA: Geração de Hash

Esta parte do código serve para a ilustração do desempenho do algoritmo SHA-256, no que diz respeito à geração de hashes, através de valores estatísticos relevantes e de um gráfico, usando ‘matplotlib’.



Os gráficos apresentados ilustram o desempenho do algoritmo SHA-256 na geração de hashes para ficheiros de tamanhos diferentes.

Observa-se uma evolução gradual e previsível do tempo de execução, com aumentos mais notáveis à medida que os tamanhos dos ficheiros crescem significativamente. Este comportamento confirma a eficiência e escalabilidade do SHA-256, especialmente em ficheiros pequenos e médios, onde o tempo de hash permanece praticamente constante e muito reduzido.

A curva ascendente nas maiores dimensões reflete a proporcionalidade do esforço computacional em relação ao volume de dados a processar, o que é esperado num algoritmo de digestão criptográfica.

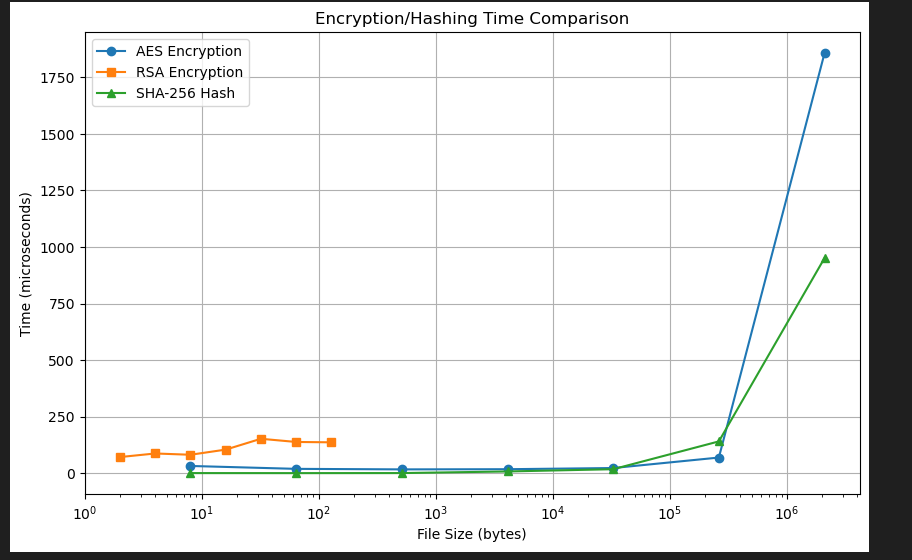
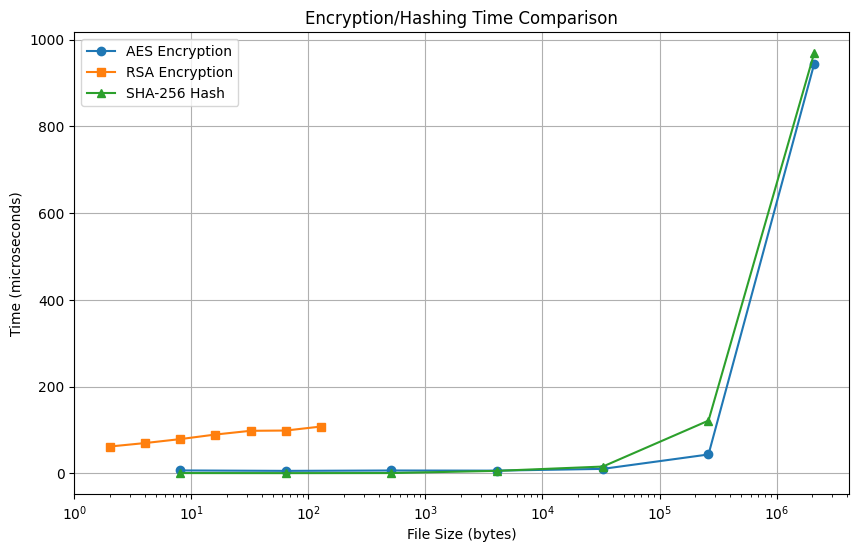
Ainda assim, mesmo nos maiores ficheiros, os tempos continuam bastante baixos, reforçando a adequação do SHA-256 para aplicações que exigem rapidez e segurança na verificação de integridade de dados.

- Comparação entre os três

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Esta parte do código serve para ilustrar o desempenho da encriptação dos algoritmos AES e RSA, bem como do algoritmo SHA-256 para geração de hashes, através de valores estatísticos relevantes e de um gráfico comparativo, utilizando ‘matplotlib’.



A análise dos gráficos revela variações evidentes no desempenho dos três algoritmos testados, quando sujeitos a ficheiros de diferentes tamanhos.

O algoritmo AES, embora mantenha uma excelente eficiência para ficheiros pequenos e médios, mostra um crescimento acentuado no tempo de execução à medida que o tamanho dos dados aumenta significativamente, no entanto, este comportamento mantém-se previsível e proporcional, o que reflete a sua boa escalabilidade mesmo em volumes elevados.

O RSA apresenta um desempenho consideravelmente inferior, com tempos de encriptação estáveis, mas mais elevados. Isto reflete a sua complexidade, por ser um algoritmo de encriptação assimétrica. O tempo de encriptação do RSA não varia de forma significativa com o aumento do tamanho do ficheiro, porque apenas uma quantidade limitada de dados pode ser diretamente encriptada com este método.

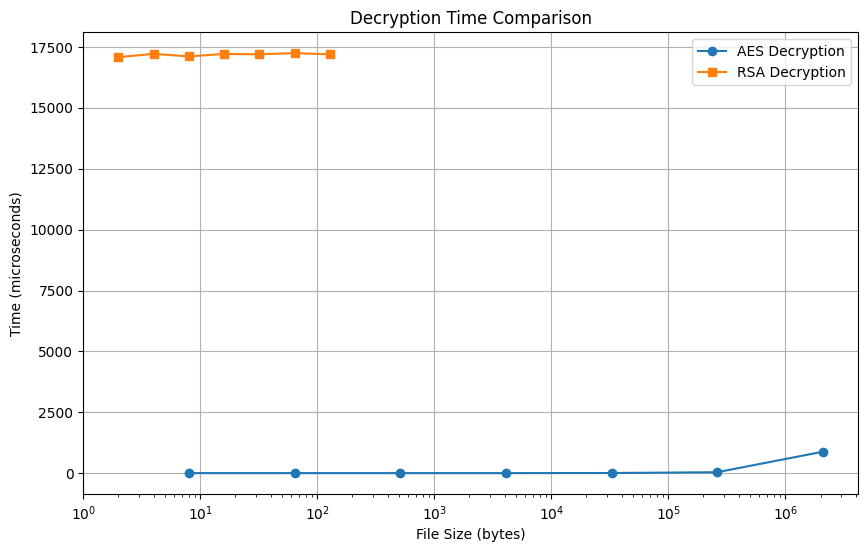
O SHA-256, sendo um algoritmo de hashing e não de encriptação, também mostra tempos bastante reduzidos para ficheiros pequenos, mas com um crescimento acentuado em ficheiros de maior dimensão. A sua curva acompanha de perto a do AES em ficheiros grandes, embora com valores ligeiramente inferiores, o que demonstra um bom compromisso entre velocidade e segurança na geração de resumos criptográficos.

Esta comparação evidencia as diferenças fundamentais entre os algoritmos: o AES destaca-se pela sua velocidade e eficiência para grandes volumes de dados, o RSA pela sua robustez, mas custo computacional elevado, e o SHA-256 pela sua adequação para verificar integridade com tempos competitivos.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Esta parte do código serve para ilustrar o desempenho da desencriptação dos algoritmos AES e RSA, através de valores estatísticos relevantes e de um gráfico comparativo, utilizando ‘matplotlib’.

Uma imagem com texto, captura de ecrã, Gráfico, file

Os conteúdos gerados por IA poderão estar incorretos.

Os gráficos apresentam a comparação direta entre os tempos de desencriptação dos algoritmos AES e RSA, evidenciando diferenças claras no desempenho de ambos.

O algoritmo AES demonstra uma subida, com tempos de desencriptação muito baixos, mesmo para ficheiros de grande dimensão. À medida que o tamanho dos dados aumenta, o tempo de execução cresce de forma gradual, mantendo-se sempre dentro de limites altamente eficientes.

Por outro lado, o RSA apresenta tempos de desencriptação bastante mais elevados e relativamente constantes ao longo dos diferentes tamanhos de ficheiro. Isto deve-se à complexidade computacional da operação de desencriptação com chave privada, que envolve cálculos matemáticos mais exigentes e demorados, independentemente da variação no tamanho dos dados encriptados.

Esta diferença reflete as naturezas distintas dos algoritmos: enquanto o AES é desenhado para rapidez e alta eficiência em grandes volumes de dados, o RSA privilegia a segurança e a troca segura de chaves, com custos de desempenho associados.

Assim, em contextos onde a velocidade da desencriptação é crucial, como no tratamento de ficheiros grandes, o AES revela-se muito mais adequado.

  - Comparação entre Ambientes de Trabalho e Configurações

No caso do AES, que depende de operações rápidas e repetidas sobre blocos de dados, notam-se diferenças claras de desempenho. Um ambiente mais otimizado consegue executar estas tarefas de forma mais eficiente, o que se reflete nos tempos de encriptação e desencriptação.

Com o RSA, como envolve cálculos pesados com números grandes, a capacidade de lidar com este tipo de operações influencia bastante os resultados, especialmente nas fases de desencriptação, onde as exigências computacionais são maiores.

Já no SHA-256, embora o algoritmo seja mais leve, ainda se observam melhorias nos tempos de execução num ambiente mais moderno. Mesmo tarefas simples acabam por beneficiar de um sistema mais eficiente.

No geral, esta comparação mostra que, mesmo com o mesmo sistema operativo e a mesma quantidade de memória, as diferenças de desempenho existem e são relevantes. A arquitetura do sistema tem um impacto real na performance em tarefas criptográficas.

Outra observação relevante foi na influência do modo de energia do sistema na execução dos testes, tendo valores mais reduzidos ao utilizar a bateria em ‘Modo de Desempenho’, possibilitando a liberação máxima do potencial do processador.

Relevância Estatística da Medição do Tempo

Para garantir a fiabilidade e a significância estatística dos resultados obtidos, realizámos uma análise mais profunda do desempenho dos algoritmos através de execuções repetidas. Esta abordagem permite verificar a consistência dos tempos de execução tanto em ficheiros fixos (iguais em todas as iterações), como em ficheiros aleatórios de tamanho constante.

Através da execução de 100 iterações por cenário, foi possível recolher uma amostra representativa de tempos para cada operação - encriptação, desencriptação e hashing. De seguida, recorremos a boxplots para ilustrar a distribuição desses tempos, destacando elementos como a média, a mediana, desvio padrão, valores mínimos e máximos e eventuais outliers. Esta visualização permite comparar o comportamento estatístico das operações realizadas sobre ficheiros idênticos e aleatórios, identificando eventuais inconsistências na execução dos algoritmos.

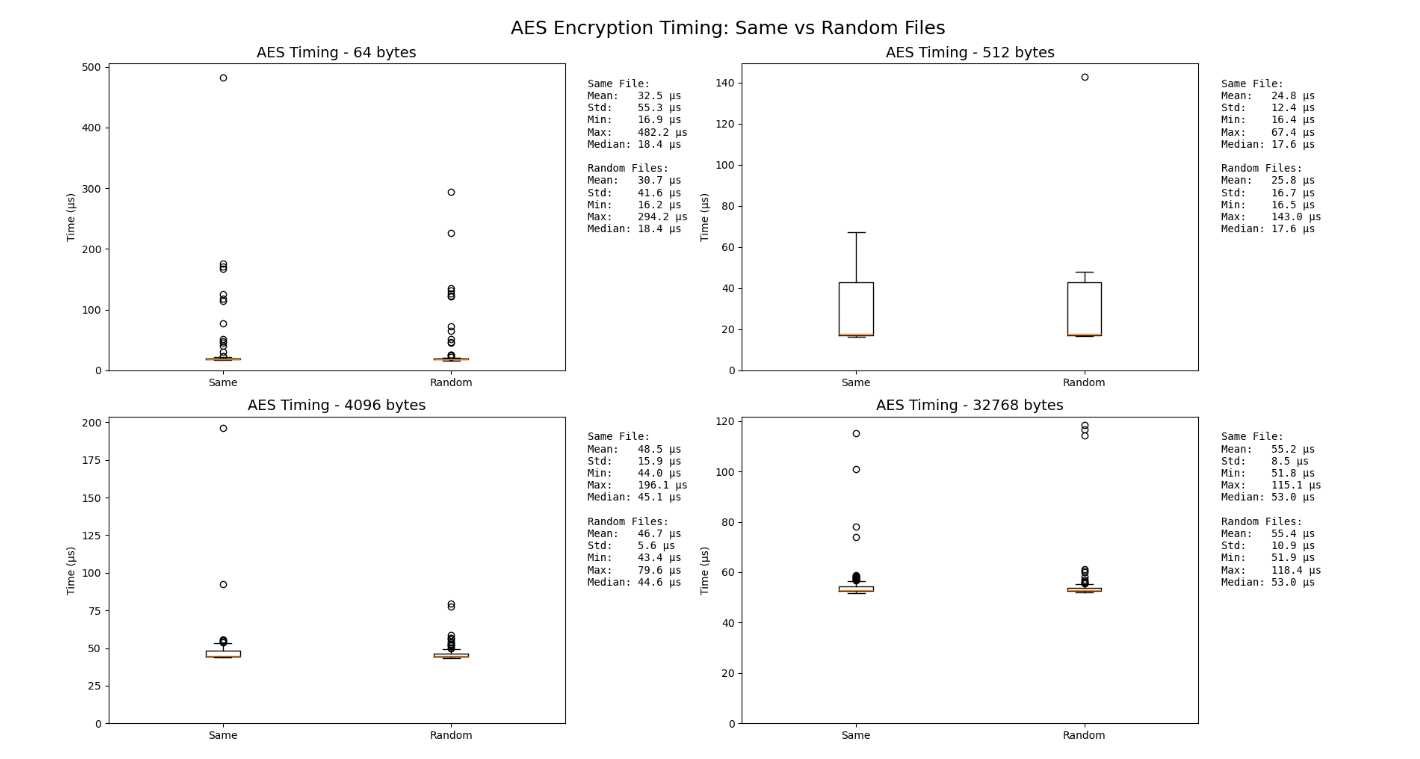
A seguir, apresentamos o código utilizado para a criação destes gráficos e os resultados.

Uma imagem com texto, captura de ecrã, ecrã, software

Os conteúdos gerados por IA poderão estar incorretos.

A criação de todos os outros boxplots são feitos de forma análoga.

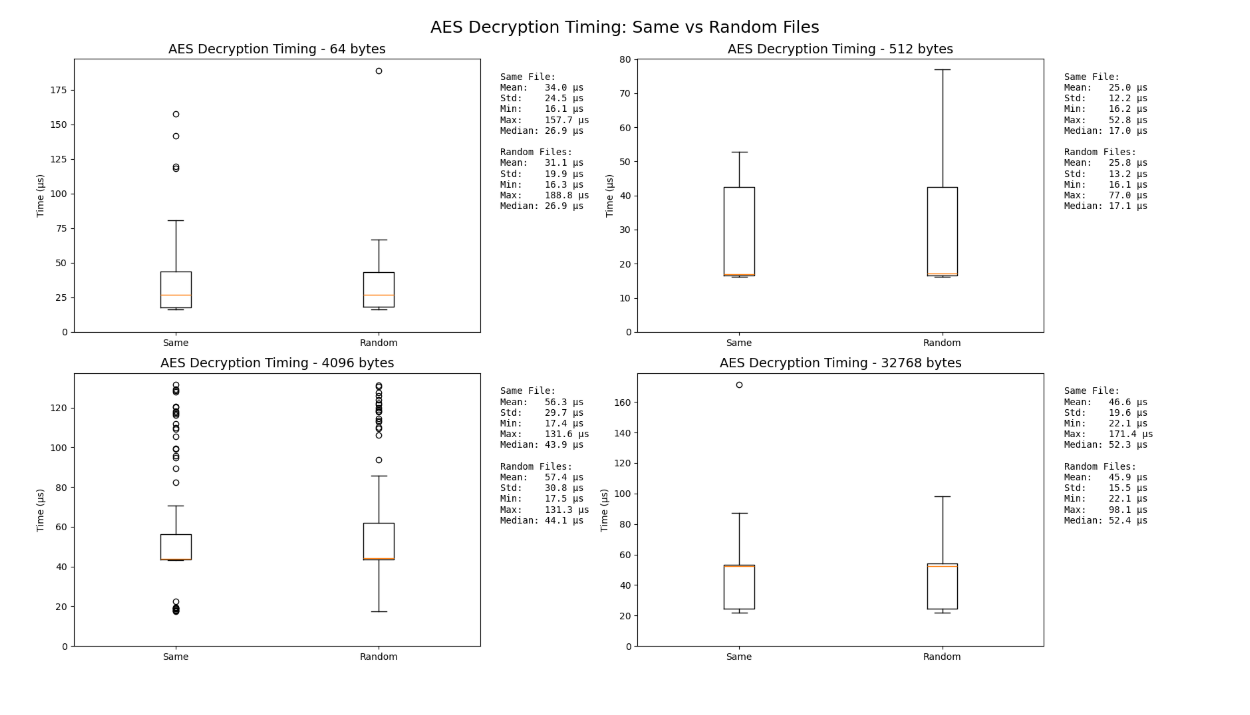
- AES: Encriptação e Desencriptação



A análise dos boxplots do tempo de encriptação do algoritmo AES mostra que os tempos de execução são bastante consistentes entre ficheiros com o mesmo conteúdo e ficheiros gerados aleatoriamente. As medianas são praticamente idênticas em todos os casos, o que indica que o conteúdo dos dados não influencia de forma significativa a performance do algoritmo.

Nota-se uma ligeira maior dispersão nos ficheiros de maior dimensão, especialmente nos dados aleatórios, com alguns outliers mais visíveis.

No entanto, estes valores extremos são pontuais e não afetam de forma relevante a tendência geral dos resultados, que confirmam a eficiência e estabilidade do AES mesmo em cenários variáveis.



Nos gráficos de desencriptação do algoritmo AES, observa-se uma consistência geral entre os tempos obtidos com ficheiros iguais e aleatórios.

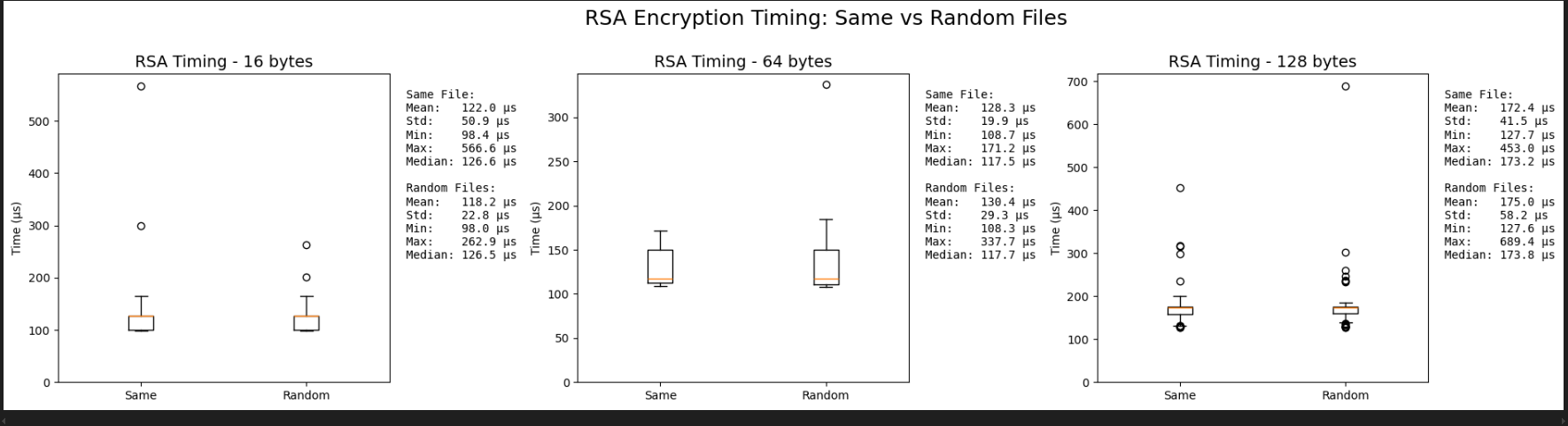
As distribuições são bastante semelhantes, com médias e medianas próximas, o que indica que a aleatoriedade dos dados não tem um impacto significativo nesta operação.

Esta estabilidade reforça a eficiência e previsibilidade do AES no processo de desencriptação.

Porém, são visíveis alguns outliers, principalmente nos ficheiros de menor tamanho, estes valores extremos podem dever-se a alterações no sistema durante a execução dos testes, mas não afetam de forma relevante a análise estatística geral.

Os dados confirmam que o AES mantém um comportamento estável e eficiente na desencriptação, independentemente da natureza dos ficheiros.

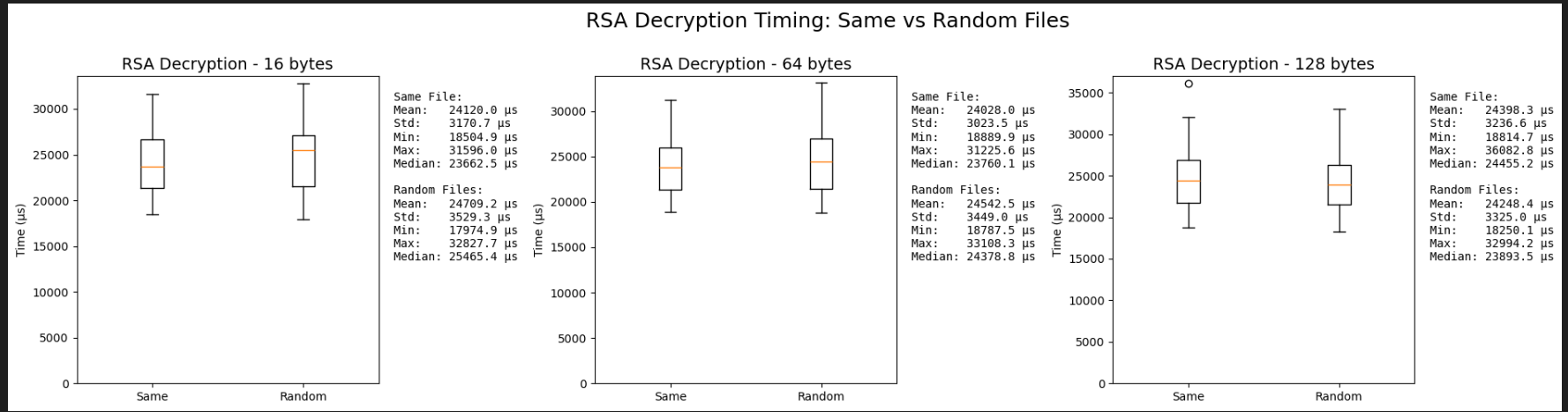
- RSA: Encriptação e Decriptação



No caso do algoritmo RSA, os gráficos mostram uma distribuição de tempos relativamente consistente entre ficheiros iguais e aleatórios, com médias e medianas muito próximas.

No entanto, observa-se uma presença notável de outliers, especialmente nos ficheiros aleatórios, o que pode indicar variações momentâneas no processamento ou interferência de outros processos no sistema.

Esta variabilidade tende a aumentar com o tamanho do ficheiro, refletindo a sensibilidade do RSA ao conteúdo dos dados e à sua estrutura binária interna.

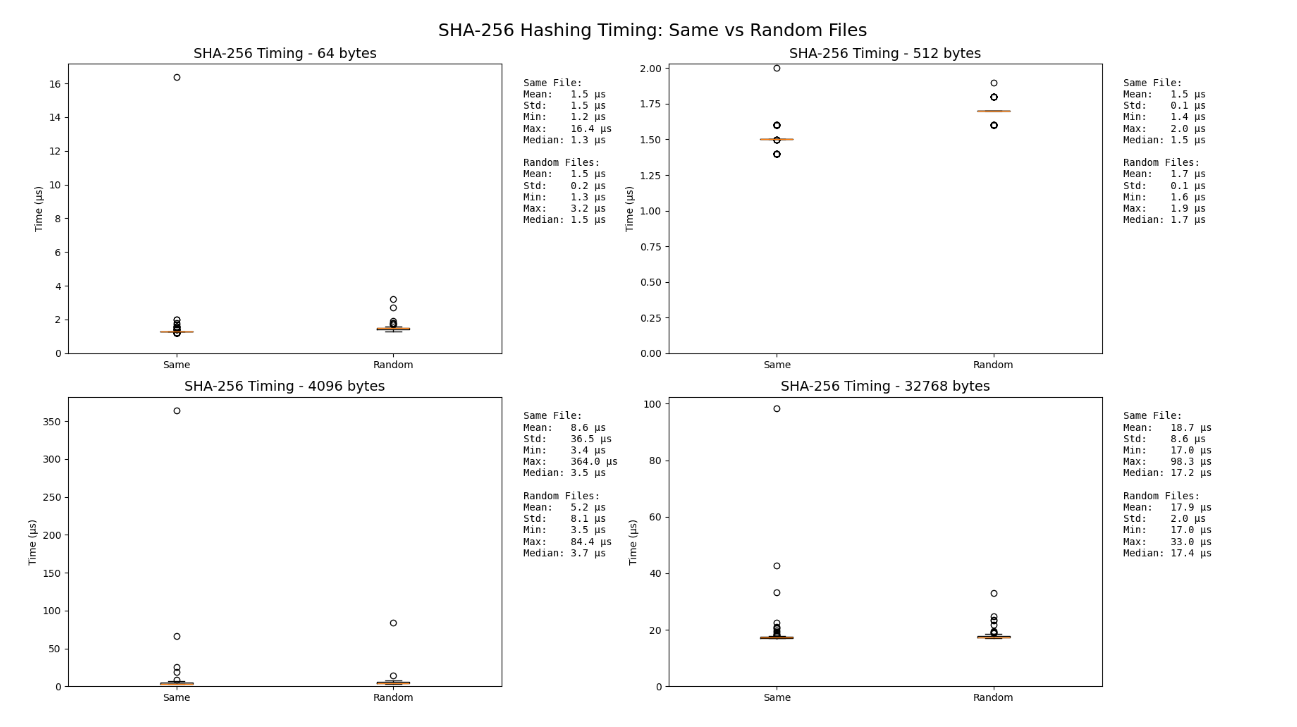


Na desencriptação com RSA, os gráficos mostram uma elevada estabilidade nos tempos de execução entre ficheiros iguais e aleatórios, com valores médios e medianos muito próximos. Isto demonstra que a natureza dos dados não tem um impacto relevante no desempenho do algoritmo nesta operação.

Ainda assim, observa-se uma maior dispersão em comparação com o AES, com a presença de outliers mais marcados, sobretudo nos testes com ficheiros de 64 bytes. Esta variabilidade pode refletir o custo computacional mais elevado do RSA e pequenas oscilações no ambiente de execução.

Apesar disso, o desempenho mantém-se consistente, confirmando a robustez do algoritmo.

- SHA: Geração de Hash



Quanto ao SHA-256, os tempos de hashing são geralmente muito baixos e estáveis, com distribuições bastante semelhantes entre os dois tipos de ficheiro.

Ainda assim, também se notam alguns outliers em tamanhos maiores, especialmente nos ficheiros iguais, o que pode ser explicado por picos momentâneos no processamento.

De forma geral, os resultados confirmam que o desempenho do SHA-256 é altamente previsível e pouco afetado pela aleatoriedade dos dados, sendo o mais consistente dos três algoritmos analisados.

Conclusão

Este trabalho permitiu-nos perceber, de forma prática e detalhada, como diferentes algoritmos criptográficos se comportam quando aplicados a ficheiros de vários tamanhos.

Avaliámos o desempenho do AES (simétrico), do RSA (assimétrico) e do SHA-256 (hash), medindo o tempo que cada um demora a processar os dados, tanto em situações repetidas como com dados aleatórios.

O AES mostrou-se o mais eficiente, com tempos de encriptação e desencriptação muito baixos e bastante consistentes, mesmo com ficheiros grandes. O crescimento do tempo é gradual e proporcional ao aumento do tamanho dos ficheiros, o que demonstra boa escalabilidade. Além disso, os tempos de encriptação e desencriptação são muito semelhantes, o que reforça a sua fiabilidade para uso em contextos que exigem rapidez e estabilidade.

O RSA, por outro lado, revelou-se muito mais pesado, principalmente na fase de desencriptação. A encriptação até se manteve estável, mas a desencriptação foi claramente mais demorada, como seria de esperar num algoritmo assimétrico. Isto mostra que o RSA não é ideal para encriptar grandes volumes de dados, mas sim para situações pontuais como a troca de chaves ou autenticação de mensagens.

No caso do SHA-256, os resultados foram também muito positivos. A geração de hashes foi rápida e consistente, mesmo com ficheiros maiores. O crescimento do tempo foi proporcional ao tamanho dos dados, e não houve grandes variações entre ficheiros iguais ou aleatórios, o que indica uma boa robustez e previsibilidade. Este algoritmo é claramente adequado para verificar integridade de dados com rapidez.

A comparação entre os três deixa claro que cada um serve um propósito diferente: o AES é indicado para encriptação contínua e rápida, o RSA para garantir segurança em comunicações específicas e o SHA-256 para verificar se os dados foram ou não alterados.

Também testámos os algoritmos em dois computadores diferentes (com o mesmo sistema operativo e RAM), e notámos diferenças nos tempos de execução. Isto mostra que o hardware tem influência real no desempenho, especialmente em operações mais exigentes como o RSA. O modo de energia (como “Modo de Desempenho”) também teve impacto nos resultados.

No geral, este projeto permitiu-nos aplicar conceitos de criptografia e também perceber, na prática, o impacto de decisões técnicas como o tipo de algoritmo, o tamanho dos dados ou até o ambiente de execução. O uso de testes repetidos e análise estatística ajudou-nos a garantir que os resultados são fiáveis e que as conclusões têm base sólida.

Referências

<https://docs.python.org/3/library/os.html>

<https://docs.python.org/3/library/timeit.html>

<https://docs.python.org/3/library/time.html>

<https://docs.python.org/3/library/statistics.html>

<https://docs.python.org/3/library/hashlib.html>

<https://docs.python.org/3/library/random.html>

<https://matplotlib.org/>

<https://numpy.org/>

<https://cryptography.io/en/latest/hazmat/>

<https://cacr.uwaterloo.ca/hac/>

<https://www.ssldragon.com/pt/blog/criptografia-aes-e-rsa/>

<https://www.splashtop.com/pt/blog/aes-encryption>