Avaliação de diferentes implementações do sistema de criptografia RSA

Ana Carla Quallio Rosa[1], **Rodrigo Campiolo**[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão – PR – Brasil

anacarlarosa@alunos.utfpr.edu.br, rcampiolo@utfpr.edu.br

Abstract. *With the increase in the flow of information between different devices, encryption systems have become essential to ensure the confidentiality and integrity of data. The RSA algorithm stands out as an effective option to provide data security. This work presents ongoing research that evaluates RSA implementations, considering different languages and cryptographic libraries. Preliminary findings, in a single programming language, already indicate significant differences in the average execution time of each encryption process.*

Resumo. *Com o aumento do fluxo de informações entre diferentes dispositivos, os sistemas de criptografia tornaram-se essenciais para garantir a confidencialidade e a integridade dos dados. O algoritmo RSA se destaca como uma opção eficaz para prover essas garantias. Este trabalho apresenta uma pesquisa em andamento que avalia implementações do RSA, considerando distintas linguagens e bibliotecas criptográficas. Os resultados preliminares, obtidos em um único ambiente de programação, já revelam diferenças significativas no tempo médio de execução de cada processo de criptografia.*

1. Introdução

Diante da expansão da área de comunicação e o aumento no fluxo de troca de informações nos dispositivos eletrônicos, os sistemas de criptografia tornam-se essenciais para manter a confidencialidade e integridade dos dados. Isso porque a criptografia é uma técnica pela qual mensagens, como texto, imagem ou outro formato, sofrem um processo de transformação em que apenas as partes autorizadas, remetente e destinatário, conseguem interpretar a informação real [Stallings and Vieira 2008].

Uma das possibilidades consiste na adoção de sistemas de chave simétricas, em que uma mesma chave é utilizada para cifrar e decifrar mensagens. Todavia, para garantir o sigilo, é preciso que a chave seja compartilhada em um meio distinto do qual realiza-se a troca de informações, o que pode ser um processo dificultoso. Nessa perspectiva, introduz-se o sistema de chave pública, também conhecido por criptografia assimétrica. Em suma, tal método possui duas chaves, uma utilizada para cifrar mensagens, podendo ser compartilhada, e outra acessível apenas ao destinatário, garantindo a exclusividade no processo de decifração. O algoritmo RSA emerge como um dos sistemas mais amplamente adotados nessa abordagem de criptografia [Imam et al. 2021].

Desde a publicação por [Rivest et al. 1978], o RSA possui aplicações em diversos campos do conhecimento científico, como a geração de assinaturas digitais, autenticação de sítios e segurança de transações. No entanto, diante de um cenário de constantes

atualizações na área de Segurança Cibernética, o algoritmo original torna-se mais propenso a ataques. Sob tal ótica, diversos trabalhos focam na melhoria do RSA, entretanto, há a necessidade de uma taxonomia consistente para alinhar requisitos como tempo de execução, desempenho e segurança [Imam et al. 2021].

O objetivo principal deste trabalho é conduzir uma avaliação de desempenho das implementações do RSA. Para tanto, investiga-se o impacto da adoção de diferentes abordagens, incluindo a seleção de bibliotecas em uma mesma linguagem, a utilização de linguagens de programação distintas e ajustes no processo matemático subjacentes à geração de chaves do algoritmo.

2. Trabalhos relacionados

Desde a publicação por [Rivest et al. 1978], diversas modificações foram propostas no sistema de criptografia RSA. [Imam et al. 2021] realizaram uma revisão sistemática da literatura, propondo uma categorização das modificações do algoritmo em subconjuntos. Nesse contexto, destacam-se as classificações em versões híbridas e alterações matemáticas.

Ao analisar cada uma dessas categorias, alguns aspectos importantes emergem. Em primeiro lugar, em relação às versões híbridas, a maioria dos estudos propõe abordagens distintas para aprimorar a segurança e a eficiência por meio da integração de propriedades de outros algoritmos. Por exemplo, [Gupta and Sharma 2012] apresentam um método que combina conceitos do RSA com o sistema *Diffie-Hellman*. [Jintcharadze and Iavich 2020], por sua vez, propõem uma abordagem híbrida do RSA com a aplicação do sistema *ElGamal*, aumentando a dificuldade de fatoração.

No que tange às modificações matemáticas, a maioria dos trabalhos explora o processo de geração das chaves. [Islam et al. 2018], por exemplo, generalizaram o RSA para n números primos, em que duas chaves públicas e privadas são produzidas, aumentando a dificuldade de fatoração e, consequentemente, o custo computacional.

Diante de um cenário de constantes atualizações em ataques cibernéticos, essas modificações são essenciais para manter o sistema de criptografia RSA seguro em diferentes aplicações. Nesse contexto, verifica-se que uma maior segurança, de modo geral, implica no aumento da complexidade computacional do algoritmo. Desse modo, torna-se necessário analisar o desempenho de diferentes implementações.

O estudo realizado por [Singh et al. 2016] comparou o desempenho do algoritmo RSA com o de curvas elípticas. Os autores conduziram uma avaliação do tempo de geração das chaves em tamanhos distintos, bem como dos processos de cifração e decifração. Os resultados apontaram que a criptografia baseada em curvas elípticas demonstrou um desempenho superior em relação ao algoritmo RSA nesse contexto de avaliação.

De modo geral, percebe-se que os estudos voltados para a avaliação do desempenho do RSA frequentemente se concentram em comparações com outros sistemas criptográficos. Contudo, não foram identificados trabalhos que exploram as possíveis variações nas implementações do algoritmo e as implicações resultantes da adoção de diferentes bibliotecas no contexto de uma mesma linguagem de programação, bem como em linguagens distintas.

3. Materiais e métodos

A Figura 1 oferece uma visão geral do método de pesquisa adotado neste trabalho em andamento.

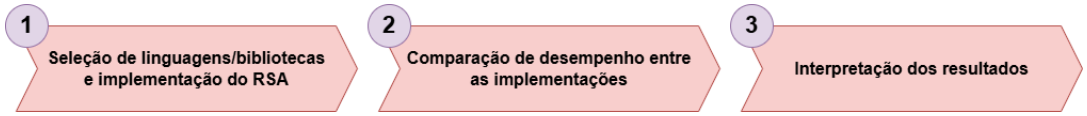

Figura 1. Método de pesquisa

A seguir, são apresentadas cada etapa do método:

1. Seleção de linguagens/bibliotecas e implementação do RSA: Esta etapa compreende a escolha de linguagens de programação e de bibliotecas criptográficas para a implementação do RSA. Escolheu-se cinco linguagens de programação, a saber: C, C++, Java, Python e Rust. Essa seleção se deu porque, segundo *rankings* de classificação, como [TIOBE 2024] e [PYPL 2024], tratam-se de linguagens mais utilizadas em cenários de diversos propósitos. Salienta-se que Rust foi escolhida porque trata-se de uma linguagem projetada para garantir a segurança de memória e concorrência. Assim, a análise das possibilidades de implementações criptográficas é de suma importância.
2. Comparação de desempenho entre as implementações: Esta etapa compreende a comparação dos processos de geração de chaves, cifração e decifração em cada biblioteca por meio de *microbenchmarking*. Desse modo, são consideradas distintas configurações de chaves e número de iterações;
3. Interpretação dos resultados: Esta etapa visa analisar e discutir os gráficos e tabelas gerados, apresentando um panorama entre as linguagens selecionadas, de modo a comparar os desempenhos obtidos.

3.1. Avaliação experimental preliminar das bibliotecas

Os experimentos iniciais foram realizados com a linguagem de programação Python, por meio das bibliotecas *Cryptography* e *PyCryptodome*, e foram conduzidos em uma máquina com processador AMD *Ryzen* 5 (64 *bit*), 12 GiB de memória RAM e sistema operacional *Debian* 12 (64 *bit*). As execuções ocorreram na interface gráfica GNOME, considerando os processos do sistema e o terminal em operação para o experimento.

Em uma linguagem de programação, há ferramentas específicas para a análise com *microbenchmarking*. No Python, tem-se o *pytest*, que permite a execução de testes detalhados de cada função do código. No experimento, foi conduzida uma avaliação que envolveu a medição do tempo de execução. Essa análise foi realizada com o *pytest* e sem a utilização de uma ferramenta específica de *microbenchmarking*, por meio das bibliotecas *timeit* e *psutil*. Isso com o intuito de comparar se os tempos obtidos são próximos. A mensagem a ser cifrada tem um tamanho de 190 *bytes*, que é o máximo permitido para o *padding* padrão de uma chave de 2048 bits da biblioteca *Cryptography*.

4. Resultados preliminares

O experimento foi dividido em duas etapas principais: a geração de chaves entre as bibliotecas e a análise dos processos de cifração e decifração, tanto com quanto sem a utilização

do *pytest*. O objetivo é avaliar se os tempos obtidos por meio de uma ferramenta de *microbenchmarking* permanecem próximos mesmo sem o uso dessa ferramenta. Para cada etapa, chaves de 2048 e 4096 bits foram geradas. Os resultados correspondem à média de cinco repetições para 10, 100, 1000 e 10000 iterações, o que possibilitou calcular o tempo médio da geração de chaves e dos processos de cifração e decifração.

No que tange à geração de chaves, a Tabela 1 apresenta os dados obtidos para as bibliotecas *Cryptography* e *PyCryptodome*, a partir de cinco repetições do experimento, em segundos, com uso do *pytest*.

Tamanho da chave	Biblioteca	Tempo Médio (s)	Desvio Padrão	Mediana
2048	*Cryptography*	0.16251	0.08277	0.13970
	Pycryptodome	0.42374	0.22434	0.41536
4096	*Cryptography*	1.04163	0,59759	0.86004
	Pycryptodome	5.21896	4.39681	4.41156

Tabela 1. Geração de chaves para as bibliotecas *Cryptography* e *Pycryptodome*

Prosseguindo para a análise do processo de cifração, a Figura 2 apresenta o tempo médio, medido em milissegundos, utilizando as bibliotecas *Cryptography* e *Pycryptodome*. Tal comparação considera os cenários com o uso do *pytest* e sem a utilização de ferramenta específica de *microbenchmarking*.

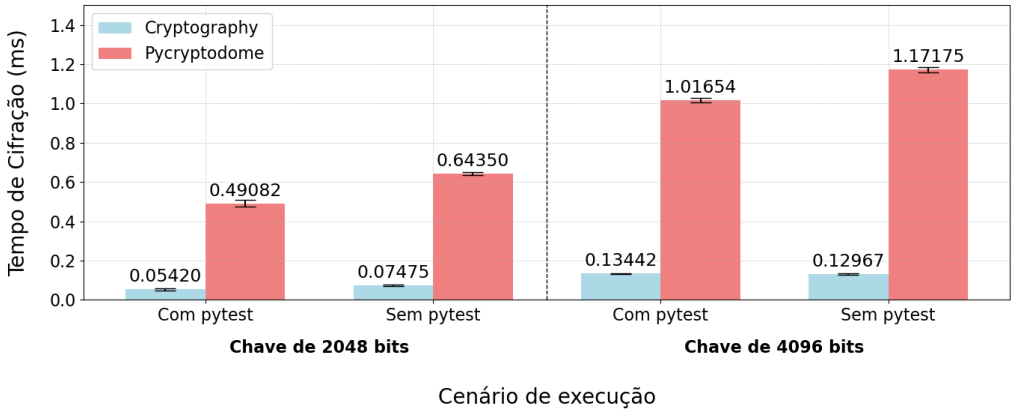

Figura 2. Tempo médio de cifração para *Cryptography* e *Pycryptodome*

Em seguida, a Figura 3 representa a comparação entre o tempo médio de decifração utilizando as bibliotecas *Cryptography* e *Pycryptodome*, com e sem a utilização do *pytest*.

De acordo com a Tabela 1 e as Figuras 2 e 3, verifica-se que a biblioteca *Cryptography* demonstrou um desempenho superior em relação à *PyCryptodome* em todas as etapas do processo de criptografia. Acredita-se que esse resultado se dá porque a *Cryptography* utiliza OpenSSL para o desenvolvimento das operações criptográficas, enquanto a *Pycryptodome* implementa a maior parte dos algoritmos em *Python*.

Ademais, os resultados obtidos com o *pytest* e sem a utilização de uma ferramenta específica de *microbenchmarking* apresentaram algumas divergências. Por fim, ressalta-se que a iteração de 10 possui um desvio padrão maior em comparação com as demais

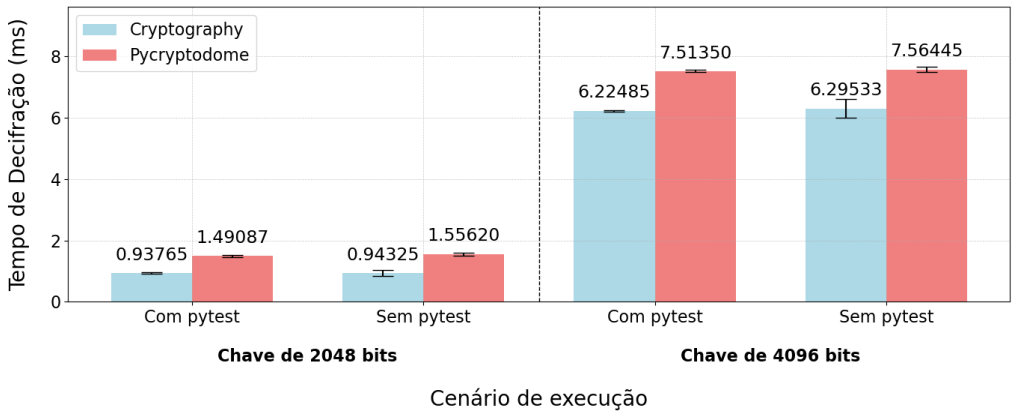

Figura 3. Tempo médio de decifração para *Cryptography* e *Pycryptodome*

iterações. Detalhes adicionais sobre os códigos do experimento e os dados obtidos estão disponíveis em um repositório no *GitHub*[1].

5. Considerações

Este artigo apresentou a investigação de diferentes implementações do RSA. Os resultados obtidos neste experimento inicial, realizados em um mesmo ambiente de programação e considerando apenas a métrica de tempo de execução, já revelam diferenças significativas entre as implementações do algoritmo. Diante disso, pretende-se prosseguir com o trabalho ao explorar outras linguagens de programação e abordagens para este sistema criptográfico, visando identificar questões de desempenho, oportunidades de otimização e avaliar os possíveis impactos na segurança diante das escolhas de projeto de cada implementação.

Referências

Gupta, S. and Sharma, J. (2012). A hybrid encryption algorithm based on rsa and diffie-hellman. In *2012 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–4.

Imam, R., Areeb, Q. M., Alturki, A., and Anwer, F. (2021). Systematic and critical review of rsa based public key cryptographic schemes: Past and present status. *IEEE Access*, 9:155949–155976.

Islam, M., Islam, M., Islam, N., and Shabnam, B. (2018). A modified and secured rsa public key cryptosystem based on “n” prime numbers. *Journal of Computer and Communications*, 6:78–90.

Jintcharadze, E. and Iavich, M. (2020). Hybrid implementation of twofish, aes, elgamal and rsa cryptosystems. In *2020 IEEE East-West Design & Test Symposium (EWDTS)*, pages 1–5.

PYPL (2024). Pypl - popularity of programming language. Disponível em: `https://pypl.github.io/PYPL.html`. Acesso em: 27 jun. 2024.

[1]Disponível em: *https://github.com/anacarlaquallio/WTICG-SBSeg*.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

Singh, S. R., Khan, A. K., and Singh, S. R. (2016). Performance evaluation of rsa and elliptic curve cryptography. In *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, pages 302–306.

Stallings, W. and Vieira, D. (2008). *Criptografia e segurança de redes: princípios e práticas*. Pearson Prentice Hall.

TIOBE (2024). Tiobe index - the software quality company. Disponível em: `https://www.tiobe.com/tiobe-index`. Acesso em: 27 jun. 2024.