Relatório RSA com Múltiplos Primos

Denise F. de Rezende - 202000560272

RSA – Rivest-Shamir-Adleman – é um sistema de criptografia utilizado para transmissão segura de dados. Existem múltiplos métodos para implementação do RSA. Nesse relatório, será analisado o RSA com Múltiplos Primos *(Multi-prime RSA)*.

---

1. Introdução

O RSA com Múltiplos Primos difere do RSA Tradicional [1] pela geração de chaves e pela decriptografia. Esse método, que visa deixar o processo de decriptografia mais eficiente, utiliza k primos para geração de chaves. A proposta para geração de chaves, conforme descrita na dissertação [2], utiliza como entrada dois parâmetros: o primeiro define o tamanho em bits do produto dos primos; e o segundo representa a quantidade k de primos.[1] Esse processo de geração de chaves produz duas chaves públicas e duas chaves privadas.

Com as chaves públicas, é feita a criptografia de uma mensagem $M \in Z_N$, sendo Z_N o anel dos naturais módulo N, o produto dos primos. O processo de criptografia de M é realizado da mesma forma que no RSA tradicional.

Para decriptografar uma mensagem criptografada, o RSA com Múltiplos Primos baseia-se no Teorema Chinês do Resto [3].

---

2. Implementação e Metodologia

A implementação para este trabalho foi feita na linguagem Python 3.9. Vale ressaltar que as bibliotecas utilizadas têm especificações que restringem o uso das mesmas. Por exemplo, no processo de geração de chaves, a escolha dos primos deve utilizar a biblioteca `secrets` ao invés da `random` com base nos critérios de segurança e criptografia como recomendado em [4]. A biblioteca `secrets` é desenhada para melhorar questões de segurança e, por esse motivo, não há informações publicadas que descrevem suas funções. Experimentalmente, pode-se observar que a função `secrets.choice(list)` tende a não escolher primos pequenos quando `list` for uma lista de primos de 3 a 999983.

---

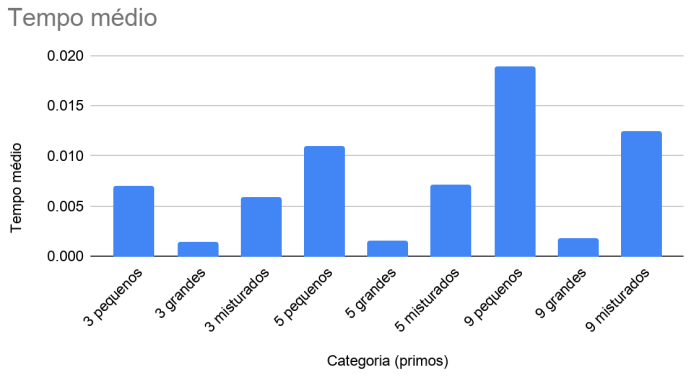
[1] Por questões de segurança, não é recomendado por [2] que se use mais de 3 primos para produtos de primos de tamanho igual ou menor que 1024 bits.

3. Resultados Experimentais

Os testes experimentais foram realizados em uma máquina Apple MacBook Pro de 16GB de RAM, com processador Intel Core I7 2.6GHz sob MacOS Mojave. Para calcular o tempo de cada execução, foi utilizada a biblioteca `time`, de forma que o intervalo de tempo medido foi o do processo de geração de chaves, geração de mensagem, criptografia e decriptografia.
Os casos testes foram separados nas seguintes três categorias: primos pequenos (3 a 1021), primos grandes (1031 a 999983) e misturados. Cada categoria foi subdividida em 3, 5 e 9 primos. Em cada uma dessas subcategorias, foram feitos, aproximadamente, mil testes.
Com os resultados obtidos de cada subcategoria, foi feita uma média do tempo de execução e outra do percentual de sucesso, que estão apresentados na tabela abaixo.

Subcategorias	Tempo médio	Percentual de sucesso
3 pequenos	0.0070029849	100
3 grandes	0.0014519230	100
3 misturados	0.0047271102	100
5 pequenos	0.0109400799	100
5 grandes	0.0015905950	100
5 misturados	0.0062070155	100
9 pequenos	0.0189332983	100
9 grandes	0.0017735972	100
9 misturados	0.0107051144	100

O seguinte gráfico mostra o tempo médio de execução para as subcategorias:



A implementação do processo de criptografia e decriptografia foi a mesma para todas as categorias, porém, a geração de chaves foi distinta. O algoritmo usado para a geração de primos pequenos e de primos grandes foi o mesmo, mas com entradas diferentes. Já para primos misturados, uma vez que era preciso escolher primos grandes e pequenos, foi necessário criar outra função para a geração de chaves. Dessa forma, por conta das características da biblioteca

`secrets`, foi necessário estabelecer dois intervalos, cada qual com um laço vinculado à obtenção de x primos na categoria primos pequenos e de y primos na categoria primos grandes.

Para avaliar os resultados, analisarei três aspectos: eficácia, eficiência e segurança.

1. Eficácia

Podemos concluir, pela terceira coluna da tabela acima, que os algoritmos foram eficazes no quesito de criptografia e decriptografia, ou seja, a mensagem inicial foi criptografada e após ser decriptografada o resultado estava correto.

2. Eficiência

Pelo gráfico apresentado, analisando a eficiência do programa para cada categoria, percebemos que, com a mesma quantidade de primos, os algoritmos mais rápidos foram: primos grandes, primos misturados, e por último, primos pequenos. Isso se deve ao aumento de tempo no processo de escolha de primos pequenos. Já que a função `secrets.choice(list)` é chamada mais vezes no caso das duas últimas categorias, devido à tendência dessa função de não escolher números primos pequenos, como foi citado anteriormente.

3. Segurança

Para avaliar o último quesito – segurança – ressalto que, para um algoritmo de criptografia ser seguro, a chave privada não pode ser dedutível a partir da chave pública. Além das chaves privadas serem armazenadas de forma segura, as públicas devem, necessariamente, ser complexas para se evitar a dedução das chaves privadas.

As chaves podem ser descritas da seguinte forma: a chave pública consiste do produto dos primos, e de um valor "e" – coprimo de phi, sendo phi o produto cujos fatores são os primos menos 1 – enquanto a chave privada consiste dos próprios primos e de um valor "d" – inverso multiplicativo de "e" em Z_{phi}.

A parte essencial de ser protegida são os primos porque sua obtenção a partir do produto dos primos – chave pública – é um procedimento sem solução eficiente conhecida a não ser que esse produto seja suficientemente pequeno para ser fácil de ser fatorado.
Como recomendado por [2], por motivos de segurança, não é aconselhado que se utilize mais de 3 primos para produtos de primos de tamanho menor ou igual a 1024 bits. Logo, sob a perspectiva das subcategorias (3, 5 e 9 primos), a quantia mais segura é 3 primos.

---

4. Implementação para cadeia de caracteres

Para implementar o RSA com Múltiplos Primos para uma cadeia de caracteres, pode-se utilizar três funções. A primeira converte a cadeia de caracteres para um número. A segunda aplica um *padding*, adicionando uma certa quantia aleatória. Essa etapa é feita para dificultar a decriptografia. A terceira utiliza o algoritmo já implementado e analisado neste relatório para criptografar o número.

---

5. Conclusão final

Tendo em vista o resultado dos testes e a análise deles feita anteriormente, minha recomendação é de que se deve utilizar apenas para primos grandes por razão de eficácia e eficiência. Mas, além disso, conforme recomendado por [2], não devem ser usados mais de três tais primos por razão de segurança.

---

6. Referências

[1] Disponível em: https://en.wikipedia.org/wiki/RSA_(cryptosystem). Acesso em 3 de abril de 2021.
[2] C. A. Monteiro Paixão, Implementação e Análise Comparativa de Variações do Criptossistema RSA, Universidade de São Paulo, Instituto de Matemática e Estatística, 2003. URL: https://www.ime.usp.br/~capaixao/Dissertacao.pdf.
[3] Ben Lynn. Stanford University. The Chinese Remainder Theorem. Disponível em: https://crypto.stanford.edu/pbc/notes/numbertheory/crt.html. Acesso em 12 de abril de 2021.
[4] Python Software Foundation. Python.org, 2001. Aviso da página inicial. Disponível em: https://docs.python.org/pt-br/3.7/library/random.html. Acesso em 12 de abril de 2021.
