



Instituto Tecnológico de Aeronáutica  
Curso de Engenharia da Computação  
Disciplina de CTC-21

Sistema criptográfico em RSA

Daniel Prince Carneiro  
Dylan N. Sugimoto  
Gabriel Adriano de Melo  
Laurival Siqueira Calçada Neto  
Thiago Filipe de Medeiros

Docente: Professor Carlos Ribeiro

São José dos Campos  
11/07/2017

## Designação de responsabilidades

Daniel:

Dylan: Pesquisa e compreensão do método RSA. Pesquisa e compreensão da correlação entre o método RSA e a matemática discreta. Redação da motivação e da metodologia do relatório que propôs o tema desse projeto. Redação do esqueleto do relatório. Redação da capa, motivação, metodologia e recursos utilizados.

Gabriel: Pesquisa do método RSA. Descrição do funcionamento do código. Avaliação e otimização do algoritmo. Análises de tempo e de implementação. Pesquisa e avaliação da segurança da implementação.

Laurival:

Thiago: Implementação do algoritmo RSA na linguagem Python. Comentários do código explicando e exemplificando o funcionamento do mesmo. Testes e análise do código implementado em relação ao espaço utilizado e tempo consumido durante execução.

---

Daniel Prince Carneiro

---

Dylan Nakandakari Sugimoto

---

Laurival Siqueira Calçada Neto

---

Gabriel Adriano de Melo

---

Thiago Filipe de Medeiros

## Motivação

Nesta Era da Informação, a humanidade está cada vez mais dependente da internet, pois a facilidade que essa tecnologia trouxe para o compartilhamento de informação mudou a forma como os indivíduos se relacionam consigo mesmo e com os outros ao seu redor. Contudo, um problema desse compartilhamento de informação é a segurança dos dados compartilhados, no sentido de evitar que agentes invasores sequestram esses dados ou informações compartilhadas. Afinal, como praticamente todos os setores da economia e da política utilizam computadores para armazenar e compartilhar informações, torna-se evidente que haja informações sensíveis ou confidenciais armazenadas na forma de bits; logo, esses computadores podem sofrer ataques cibernéticos, como ocorreu recentemente, no mês de Maio e Junho de 2017, em que “*hackers*” sequestraram informações de computadores de empresas e pediram um valor de resgate dessas informações para essas empresas.

Assim, nesse contexto da segurança da informação, destaca-se a criptografia RSA por possibilitar a comunicação segura de informações e, pelo menos até agora, ter resistido a todas as tentativas de quebra. Além disso, é o principal método de criptografia utilizado na segurança de dados compartilhados via internet, e por tudo isso foi escolhido como tema para este trabalho aplicar a criptografia RSA em textos em português, bem como pelo fato desse método de criptografia utilizar conhecimento embasado na Teoria dos Números, que está inserida no contexto da matemática discreta, ou seja, este tema está correlacionado com o conteúdo ministrado na disciplina CTC-21. De forma mais específica, foi utilizado conceitos da aritmética módulo  $n$  para aplicação do método RSA<sub>[1][3]</sub>, porém é sabido que a criptografia faz uso de teorias mais avançadas da matemática discreta, como por exemplo, um problema da computacional da criptografia é encontrar vetores curtos em reticulados modulares, em que nesse caso faz uso de uma teoria de reticulados.<sub>[4][5]</sub>

## Metologia

Este projeto foi executado em três etapas:

1. Busca por informações no sentido de compreender o funcionamento do algoritmo do método de criptografia RSA e a sua correlação com a disciplina de CTC-21.
2. Implementação do método em código utilizando as linguagens recomendadas, em especial, a linguagem python. E documentação do código.
3. Análise do código e redação do relatório.

Na primeira etapa, percebeu-se que há uma grande quantidade de materiais sobre a metologia RSA, e de diferentes formatos, ou seja, há diferentes fontes sobre essa metologia disponíveis na internet que variam desde web enciclopédias e outros sites da área de criptografia e matemática discreta até monografias e artigos científicos, e videoaulas sobre a metologia RSA. Diante dessa grande variedade de fontes, deu-se prioridade para as fontes com referências confiáveis, ou para as documentações advindas de instituições de educação e de pesquisa de conceito e de alto prestígio. Assim, consultando essas fontes compreendeu-se o algoritmo de geração, de encriptação e de decriptamento, e que tal método fundamenta-se na matemática módulo  $n$ .<sup>[1][3]</sup> E com uma pesquisa mais aprofundada, no sentido, de investigar mais a fundo a correlação entre a matemática discreta e a criptografia, descobriu-se a aplicação da teoria de reticulados em criptografia, em específico, na resolução do problema do vetor mais curto.<sup>[4][5]</sup>

De forma resumida, o método RSA é composto por geração das chaves, encriptação da mensagem e decriptamento da mensagem. Na geração das chaves, primeiramente escolhe-se dois números primos e realiza-se o produto deles, obtendo-se o primeiro número da chave pública e da chave privada. O segundo número da chave pública é escolhido de forma que seja primo com a função totiente do primeiro número, e seja menor que a função totiente do primeiro número. O segundo número da chave privada é tal que o produto entre o segundo número das duas chaves deixa resto unitário na divisão pela função totiente do primeiro número da chave pública. Na encriptação, cada caractere é transformado para um número de dois dígitos, e o texto é quebrado em blocos, chamados de mensagem. O número criptografado na mensagem é tal que deixa o mesmo resto que a mensagem elevada à potência do segundo número da chave pública na divisão pelo primeiro número da chave pública. No decriptamento, recupera-se a mensagem ao calcular o número que deixa o mesmo que a mensagem criptografada elevada à potência do segundo número da chave privada na divisão pelo primeiro número da chave privada. Essa lógica matemática foi implementada computacionalmente na etapa seguinte.

Na segunda etapa, implementou-se o método RSA em linguagem Python começando pela geração das chaves, depois, a encriptação de texto, e finalmente, o decriptamento, sendo que a documentação do código foi feita à medida que parte era completada. Após o término de cada parte do código (geração, encriptação e decriptamento), testes foram realizados para assegurar a funcionalidade do código. E ao fim das três partes estarem completas, iniciou-se os testes finais para assegurar o funcionamento global do código sendo seguidos pelos testes de análise de desempenho. Os resultados desses teste são descritos na secção “Resultados e Análise” deste relatório.

Na terceira etapa, foi feito uma análise do código, e melhorias foram sugestionadas e implementadas. E o projeto foi documentado na forma do presente relatório.

## Recursos e Avaliação

Neste projeto, foi utilizado as seguintes ferramentas de pesquisa e codificação:

- Pesquisa na internet;
- Consulta a materiais acadêmicos relacionados ao assunto (artigos científicos, monografia, livros, etc);
- Codificação em Python;
- Código redigido utilizando o programa “Sublime Text”.
- Redação do relatório no programa “LibreOffice Writer”;
- Geração de primos no programa “OpenSSL”

A avaliação da implementação do sistema criptográfico baseado no método RSA para textos em português deu-se na análise do tempo de processamento, da capacidade de criptografar mensagens com caracteres especiais e da segurança contra ataques de força bruta.

A análise de tempo foi feita utilizando-se recursos do próprio python, contabilizando-se os tempos para cada etapa de execução do programa.

## Implementação

O código foi dividido em três arquivos de forma a organizá-lo melhor. No arquivo “RSA.py”, mostra-se um exemplo de criptografia e descriptografia de uma mensagem de texto presente no arquivo “mensagem.txt”. No arquivo “classes.py” utilizou-se orientação a objetos para encapsular o código que definem o mensageiro e o destinatário da mensagem. No arquivo

“suporte.py” definiram-se funções matemáticas básicas e computacionais auxiliares, como o máximo divisor comum entre dois números ou ainda o inverso modular.

Deve-se utilizar dois primos maiores com mais de 100 dígitos decimais para que o código possa funcionar, apesar de que também foi definida uma função que gera primos aleatórios, mas que não é utilizada em toda a execução por ser lenta. Além da utilização de primos gerados por implementação própria, obtendo-se uma chave de 768 bits, também gerou-se primos com o programa OpenSSL, de forma a se obter uma chave de 2048 bits.

No arquivo “RSA.py”, as chaves públicas e privadas são geradas a partir dos números primos, isto é, primeiro faz-se a multiplicação de ambos os primos para obter a chave pública e o seu expoente é obtido como o coprimo do produto dos dois primos menos um. O expoente da chave privada é obtido como o inverso modular do expoente da chave pública. Em seguida, a mensagem é lida de um arquivo e criptografada utilizando-se a chave pública, sendo a mensagem criptografada impressa na tela. Por fim, a mensagem é descriptografada utilizando-se a chave privada, sendo então impressa na tela.

É importante notar que como a criptografia ocorre em texto, é necessário que o mesmo seja convertido para uma forma numérica. Para melhorar a segurança da implementação, cada grupo de 78 caracteres são convertidos para um único número, este número pode ser aumentado caso se escolham números primos ainda maiores. É importante, contudo, que a codificação da mensagem original esteja no formato UTF-8, pois cada caractere poderá ser codificado com até 1000 códigos diferentes, na implementação utilizada em python, o que cobre todos os acentos e caracteres especiais utilizados em português. Esse passo de converter o texto em número é realizado pela função “conversor” presente na classe “mensageiro”.

Após ter a mensagem de texto expressa como um vetor de números, cada número é então criptografado utilizando-se a chave pública, isto é, é realizada uma exponenciação modular desse número na base da chave pública.

Assim, a mensagem criptografada, que corresponde a esse vetor de números criptografados pode ser transmitida com segurança para o destinatário, que ao recebê-la, utilizará a sua chave privada para descriptografar cada número desse vetor de números. Assim, com um vetor de números descriptografados, basta converter cada número em uma sequência de 78 caracteres, que poderá ser lido no formato UTF-8.

## Resultados e Análise

A implementação do algoritmo teve um tempo linear para a quantidade de caracteres criptografados, o que pode ser observado nas Figuras 1 e 2. Isso se deve ao fato de que a criptografia empregada se deu por blocos de caracteres, e não pela a mensagem como um todo. Dessa forma, como cada bloco tem um tempo de processamento aproximadamente constante, o tempo de execução do algoritmo depende apenas da quantidade de blocos de caracteres a serem criptografados, e como cada bloco contém um número constante de caracteres, o tempo de execução do algoritmo depende linearmente da quantidade de caracteres a serem criptografada.

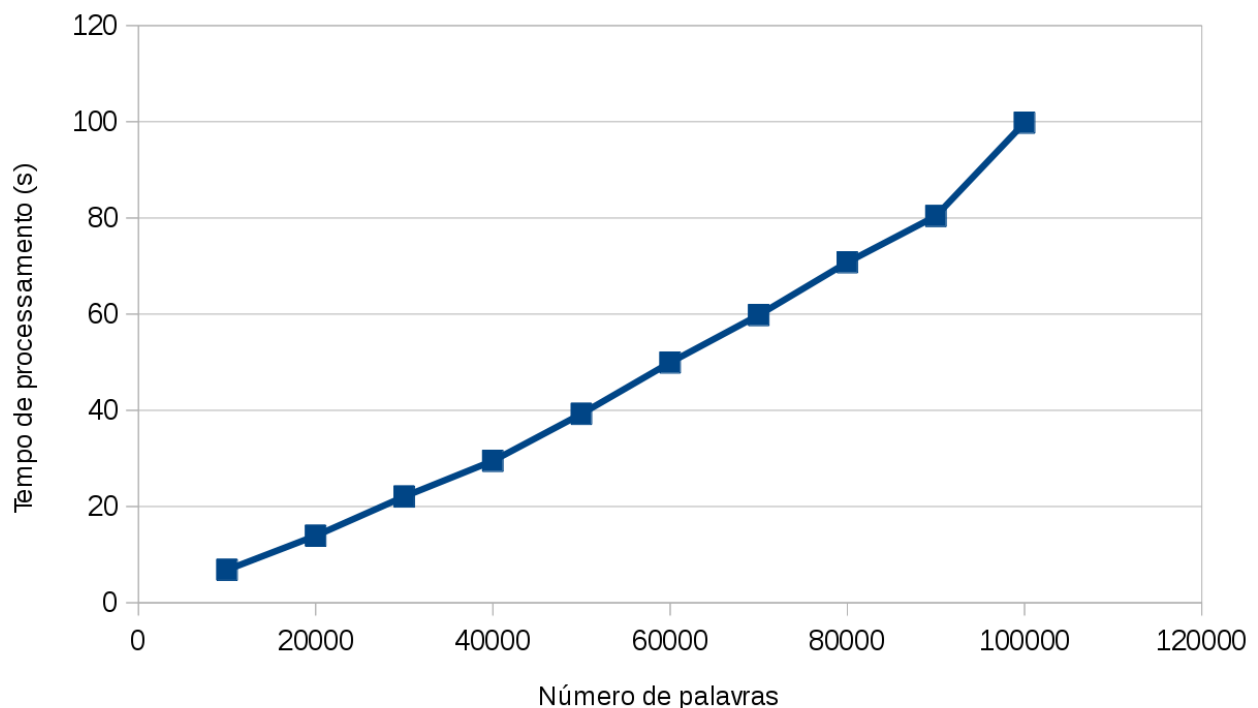


Figura 1 – Tempo de processamento em função do número de palavras.

É importante notar que cada bloco de caractere deve ter um tamanho considerável, caso contrário, um atacante que conhecesse a chave pública poderia simplesmente gerar todas as combinações de blocos possíveis. Por exemplo, considerando o caso no qual cada caractere fosse individualmente criptografado, um atacante poderia simplesmente gerar todos os 255 caracteres possíveis e assim teria a tabela numérica de transformação do caractere para o número criptografado e assim poderia descobrir a mensagem criptografada.

A segurança da implementação dá-se pelo tamanho do bloco de caracteres e pelo tamanho dos números primos empregados. O tamanho do bloco de caracteres é importante para impedir que um atacante, por força bruta, obtenha todas as possíveis combinações de blocos e seus respectivos números criptográficos e, uma vez que o processo de criptografia é determinístico, obter a mensagem original. Os grandeza da chave empregada, isto é, dos números primos, é importante não

só por determinar o tamanho máximo do bloco, como também por ser necessário fatorá-lo para obter a chave privada e poder ler a mensagem original.

A fim de avaliar a segurança do método RSA pela fatoração da chave pública, efetuou-se uma pesquisa bibliográfica. A maior chave encontrada a qual foi possível ser quebrada tinha um tamanho de 768 bits, consideravelmente próximo á chave de 781 bits utilizada inicialmente no projeto. Em um processador de um único núcleo, AMD Opteron, operando a 2.2 GHz e utilizando 2 GB de RAM, essa fatoração levaria cerca de 1500 anos [6].

Outro fator importante de segurança é o fato de que uma pequena variação na mensagem, seja de apenas um caractere, afeta significativamente o código criptografado do seu respectivo bloco de criptografia. Dessa forma um atacante não consegue avaliar, em um ataque de força bruta, se a sua tentativa está próxima da mensagem interceptada.

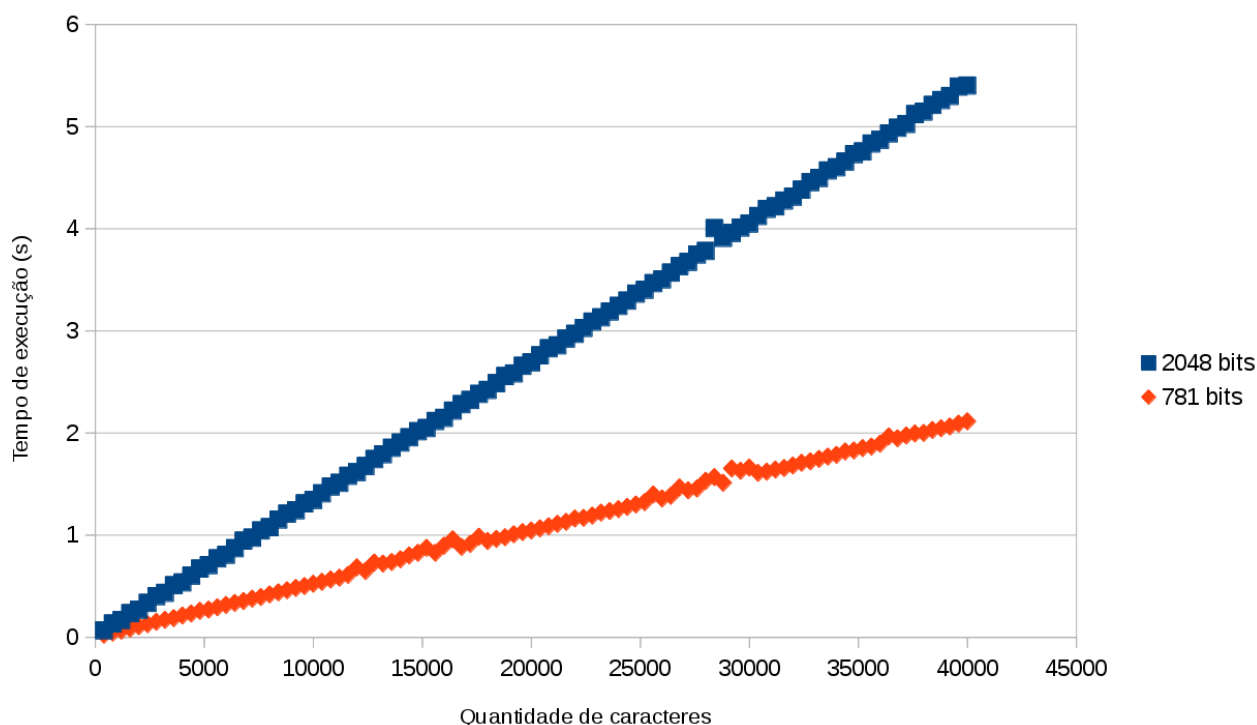


Figura 2 – Comparação entre os tempos de execução de chaves de 2048 e 781 bits.

Para gerar uma chave de 2048 bits aleatória, utilizou-se o programa OpenSSL para gerar os números primos aleatórios. Dessa forma, efetuou-se os testes com o algoritmo utilizando-se uma chave de 2048 e blocos de 250 caracteres. Como cada caractere foi representado por 8 bits na codificação utilizada, uma codificação de 2048 bits permite blocos de até 256 caracteres para que a mensagem fosse descriptografada unicamente.

Dessa forma, a implementação do algoritmo pode ser avaliada também utilizando-se primos ainda maiores do que os utilizados inicialmente. Tais primos, apesar de aumentarem a segurança contra fatoração e força bruta, aumentam o tempo de processamento, como observado na Figura 2.



Com relação aos tempos de execução, observou-se que entre os tempos de conversão e de desconversão de texto para número, o último é consideravelmente mais lento que o primeiro, o que pode ser explicado pela realização da operação módulo, presente apenas no último. Dessa forma, a operação de exponenciação modular, que realiza várias vezes a operação de módulo, é ainda mais lenta, o que explica os maiores tempos de processamento na encriptação e decriptação em todo o algoritmo.

## Considerações Finais

A implementação do sistema criptográfico baseado no método RSA para textos em português foi bem sucedida ao realizar a criptografia de caracteres especiais, ao ter uma ordem linear e a utilizar números primos e blocos suficientemente grandes para inviabilizar a sua quebra por força bruta.

Observou-se que a maior parte do tempo de processamento da implementação se deu na encriptação e na deciptação das mensagens, o que pode ser justificado pela da exponenciação modular. Uma melhoria do algoritmo seria na otimização dessa função matemática, diminuindo assim o tempo de processamento.

A fim de melhorar a segurança da implementação cada bloco de caracteres poderiam ter pelo menos um espaço destinado a um caractere aleatório, o que deixaria a mensagem criptografada não determinística. Dessa forma, não seria possível descobrir a mensagem por força bruta, uma vez que a mesma mensagem teria um código criptografado diferente. Para o destinatário bastaria descriptografar a mensagem e retirar o caractere aleatório, presente em uma posição conhecida, como no final de cada bloco.

## Referência Bibliográfica

- [1] de Oliveira, P. E. R.; de Andrade, P. T. E.; D'Oliveira, R. L. G. *Rsa*. Disponível em: [http://www.ime.unicamp.br/~ftorres/ENSINO/MONOGRAFIAS/oliv\\_RSA.pdf](http://www.ime.unicamp.br/~ftorres/ENSINO/MONOGRAFIAS/oliv_RSA.pdf), acessado em: 11 julho. 2017.
- [2] Disponível em: [https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o\\_totiente\\_de\\_Euler](https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_totiente_de_Euler), acessado em: 11 julho. 2017.
- [3] Disponível em: <https://pt.wikipedia.org/wiki/RSA>, acessado em: 20 maio. 2017.
- [4] Zanon, G. H. M.. *Criptografia Homomórfica*. São Paulo: Novembro de 2016. Disponível em: <http://bcc.ime.usp.br/tccs/2016/gustavozanon/monografia.pdf>, acessado em: 20 maio. 2017.
- [5] Bollauf, M. F. *Criptografia baseada em Reticulados*. Campinas: 21 de Novembro de 2014. Disponível em: <http://www.ime.unicamp.br/~campello/geometria/maiara.pdf>, acessado em: 20 maio. 2017.
- [6] Kleinjung et. al.; Factorization of a 768-bit RSA modulus; Disponível em: <<https://eprint.iacr.org/2010/006>>. Acessado em 4 de julho de 2017.

## Apêndice