UMA APRESENTAÇÃO SUCINTA DO SISTEMA DE CRIPTOGRAFIA RSA

Gabriel Estevam de Oliveira[1], Álvaro Junio Pereira Franco[2]
[1]Universidade Federal de Santa Catarina/Graduando de Engenharia de Computação/gabriel.estevam@grad.ufsc.br
[2]Universidade Federal de Santa Catarina/Doutor em Ciência da Computação/alvaro.junio@ufsc.br

Resumo: *Este artigo propõe o ensino do sistema de criptografia RSA em uma discipllina do curso de Engenharia de Computação da Universidade Federal de Santa Catarina (UFSC). Essa disciplina é chamada de Projeto e Análise de Algoritmos. O sistema de criptografia RSA foi desenvolvido em 1978 por Rivest, Shamir e Adleman. Este sistema é atualmente utilizado, por exemplo, nos protocolos ssh, ssl, e assinatura digital. O estudo deste sistema de criptografia proporcionou uma discussão com relação a uma maneira de manipular números inteiros grandes em um computador e uma discussão de como podemos mapear os caracteres de uma mensagem para números inteiros. Além disso, os algoritmos podem ser implementados sem dificuldades por alunos que conhecem uma linguagem de programação e estruturas de dados básicas. Os assuntos necessários para o sucesso da aprendizagem são operações comuns da aritmética básica e modular, o algoritmo estendido de Euclides e a geração de números primos grandes. O tempo de execução de alguns algoritmos serão analisados. Como resultados, obteve-se o sistema de criptografia implementado e dentre os objetivos satisfeitos a boa aceitação dos alunos com a forma que os conteúdos da disciplina foram ministrados e a experiência adquira pelos mesmos.*

Palavras-Chave: *Criptografia, RSA, Análise de Algoritmos, Implementação.*

1 INTRODUÇÃO

O tema do 6º. SICT-SUL, *A matemática está em tudo*, foi o incentivo a escrever sobre a aplicação de um assunto ligado à Matemática em uma disciplina da Computação chamada de *Projeto e Análise de Algoritmos*. Em 2016, foi apresentado à turma de *Projeto e Análise de Algoritmos* da UFSC o sistema de criptografia chamado RSA. Os sistemas de criptografias são muito importantes para a segurança e privacidade dos dados. A troca de dados entre usuários da internet, a proteção de senhas e de cartões magnéticos são algumas aplicações cotidianas. Em caso de interceptações maliciosas, os esquemas de codificações protegem a integridade dos dados e garantem a transmissão segura das informações. Uma das muitas aplicações dos sistemas de criptografia está na solução do problema da troca segura de mensagens.

Os *sistemas de criptografia com chave pública* são uma opção como solução para o problema da troca segura de mensagens. O sistema de criptografia RSA [Rivest et al. 1978], foi desenvolvido por R. L. Rivest, A. Shamir, L. Adleman no *Massachesetts Institute of Tecnology*. Este sistema é usado nos protocolos ssh (*veja em https://linux.die.net/man/1/ssh*), ssl (*veja em https://linux.die.net/man/3/ssl*), assinaturas digitais [Rivest et al. 1978], etc.

Este trabalho mostra os algoritmos dados em sala de aula, suficientes para uma implementação do sistema de criptografia RSA. O principal resultado deste trabalho é uma implementação (feita na linguagem Java) do sistema RSA. A implementação é livre e disponibilizada se requisitada. Em seguida, apresentam-se os algoritmos e suas análises.

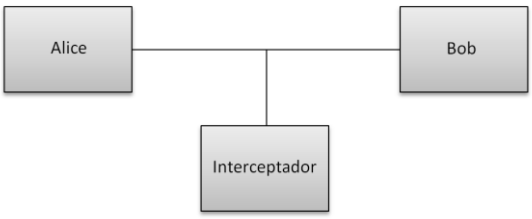
2 METODOLOGIA

A subseção a seguir descreve de maneira sucinta os algoritmos do sistema de criptografia RSA que usa a metodologia de chave pública.

2.1 A metodologia de Chave Pública do sistema RSA

Apresenta-se o seguinte problema: Alice e Bob desejam trocar mensagens por um meio público, porém de forma sigilosa. Caso alguém às intercepte, as mensagens não poderiam ser lidas porque estariam codificadas (veja Figura 1). No sistema RSA, Alice e Bob possuem cada um, uma chave pública (conhecida por todas as pessoas) e uma chave privada (conhecida somente por eles próprios – cada um conhece a sua). Se a Alice deseja enviar uma mensagem para Bob, então ela deve utilizar a chave pública de Bob para criptografar a mensagem. Assim, somente Bob conseguirá decriptar a mensagem, através da sua chave privada e, portanto, ter acesso ao conteúdo da mensagem.

Figura 1. Cenário de troca de mensagens



Fonte: produzida pelos autores.

Os caracteres das mensagens podem ser mapeados para números inteiros utilizando uma tabela de caracteres como a UTF-8. A concatenação dos números gera um número inteiro que pode ser absurdamente grande (depende do tamanho da mensagem).

Para definir as chaves públicas e privadas, considere um inteiro positivo N. O sistema RSA trabalha com números no módulo N. Um número inteiro a no módulo N é o resto da divisão de a por N. A proposta do sistema RSA é escolher dois números primos quaisquer, p e q. A chave pública será formada por dois números inteiros positivos

- $N = pq$, o produto dois números primos p e q; e
- c, um *primo relativo*[1] do produto $(p - 1)(q - 1)$.

E a chave privada é um número inteiro

- d, o *inverso multiplicativo*[2] no módulo $(p - 1)(q - 1)$ de c.

Para criptografar m, basta fazer m elevado a c-ésima potência no módulo N.

[1] Dados dois números inteiros a e b, a é *primo relativo* de b se o máximo divisor comum de a e b é 1.
[2] Pode ser obtido através do Algoritmo estendido de Euclides, para mais informações sugere-se a consulta de “Algoritmos” de S. Dasgupta, C. Papadimitriou e U. Vazirani [Dasgupta et al. 2009].

$$m' = m^{c} \pmod{N}$$

E para decriptar *m'*, basta fazer *m'* elevado a *d*-ésima potência no módulo *N*.

$$m'' = \left(m'\right)^{c} \pmod{N}$$

A prova de que *m* é igual a *m''* pode ser encontrada em [Dasgupta et al. 2009].

3 RESULTADOS E DISCUSSÃO

O principal resultado deste trabalho é uma implementação do sistema de criptografia RSA disponibilizada sempre que requisitada. No entanto, é descrito em seguida uma forma de como tratar números inteiros grandes em um computador. Durante o desenvolvimento deste trabalho, foi considerado os números estarem na base 10. A implementação foi feita em linguagem de programação Java e é código aberto.

3.1 O manuseio de números grandes

Tanto as chaves pública e privada quanto as mensagens trocadas são representadas como números inteiros grandes. Portanto, é necessário a utilização de uma estrutura de dados apropriada. A presente proposta sugere uma classe com os seguintes atributos: um vetor, que armazena cada dígito do número inteiro; o sinal (positivo ou negativo); e, a quantidade de dígitos. Os métodos sugeridos da classe são descritos em seguida.

3.2 Aritmética Básica de Números Grandes

O sistema RSA utiliza operações básica sobre números grandes como, por exemplo, somar, subtrair, multiplicar e dividir. Uma nota importante, presente em todas as operações, é quanto ao tamanho e o sinal dos números resultantes. Ao final das operações deve-se indicar o tamanho do número resultante e estabelecer o sinal do mesmo conforme a regra dos sinais de cada operação.

O método *soma* recebe dois números inteiros *a* e *b* (ambos com *n* dígitos) e devolve um número *c*, resultado da operação. No algoritmo implementado, percorre-se cada par de dígitos dos números *a* e *b* (da direita para a esquerda) somando-os e armazenando na posição corresponde em *c.* Caso a adição do par seja maior que o valor da base numérica adiciona-se o “dígito extra” ao próximo par de dígitos. No final do processo, *c* pode ter ou *n* ou *n* + 1 dígitos. O tempo de execução desta operação é *O(n)*

pois existem *n* pares de dígitos e cada par de dígito é somado juntamente com o "dígito extra" uma única vez.

O método *subtração* recebe dois números inteiros *a* e *b* (ambos com *n* dígitos) e devolve um número *c*, resultado da operação. Um método auxiliar que determina qual dos números é maior em valor absoluto é usado, para assim, subtrair o menor do maior. Percorre-se cada par de dígitos de *a* e *b* e realiza-se a subtração entre os dígitos. Se o resultado for menor que zero inverte-se o sinal e diminui-se em um o valor do próximo algarismo de *c*. No final, o sinal de *c* será igual ao sinal do maior número em valor absoluto (*a* ou *b*). O tempo de execução desta operação é *O(n)* por um motivo similar ao do método *soma*.

O método *multiplicação* recebe dois números inteiros *a* e *b* com *n* dígitos cada e devolve um número *c*, resultado da operação. Para cada dígito de *b*, começando do primeiro à direita, realiza-se a multiplicação deste dígito com *a*, somamos este resultado a um novo número (inicialmente nulo), que será o resultado final da operação. As somas devem ser deslocadas a quantidade de dígitos correspondentes ao dígito de *b* em questão. A multiplicação de um dígito de *b* com todo o *a* é feito percorrendo *a* e realizando o produto entre cada par. Caso o resultado seja maior que o valor da base numérica, guarda-se o segundo dígito para somar ao próximo dígito do número resultante. No final do processo, *c* terá no máximo *2n* dígitos. O tempo de execução deste algoritmo é $O(n^2)$ pois cada um dos *n* dígitos de *b* é multiplicado por todos os *n* dígitos de *a*.

O método *divisão* recebe dois números inteiros positivos *a* e *b*, ambos com *n* dígitos, e devolve dois inteiros *q* e *r*, onde *q* é a parte inteira da divisão de *a* por *b*, e *r* é o resto. O algoritmo implementado pode ser visualizado na figura 2. [Dasgupta et al. 2009]

Figura 2. *Divisão* em Pseudocódigo

```
DIVISÃO(a, b)
 1  se a possui um único dígito
 2     então (q, r) ← (0, a)
 3     senão armazene em d o último dígito de a
 4           desloque a à direita ▷ o último dígito de a é removido
 5           (q, r) ← DIVISÃO(a, b)
 6           desloque q e r à esquerda ▷ o dígito 0 é adicionado no final de q e r
 7           r ← r + d
 8           se r > b
 9             então encontre o maior inteiro k (entre 1 e 9) tal que r − kb ≥ 0
10                   r ← r − kb ▷ subtração de números inteiros grandes
11                   q ← q + k
12           devolva (q, r)
```

Fonte: produzido pelos autores

Existem *n* chamadas recursivas, cada uma consome um tempo da ordem de *O(n)* pois é realizado deslocamento (de dígitos), adição, subtração e multiplicação de um dígito por um número inteiro. Portanto, o tempo de execução deste algoritmo é $O(n^2)$.

3.3 Aritmética Modular

Devido ao sistema RSA trabalhar com números inteiros dentro de um determinado intervalo é necessário implementar métodos para algumas operações da aritmética modular[3].

O método *módulo N* recebe dois números inteiros positivos *x* e *N*, e devolve *x* módulo *N*. Pode-se utilizar o método *divisão* visto anteriormente para dividir *x* por *N. O r* devolvido pelo método *divisão* é o resto da divisão e também *x* módulo *N*. Logo, a análise de tempo de execução é a mesma para o método *divisão*, $O(n^2)$.

O método *multiplicação modular* recebe dois números inteiros *a* e *b*, um número inteiro *N* e devolve *ab* módulo *N*. Realiza-se a multiplicação entre *a* e *b* e encontra-se a resposta em módulo *N*. Para *multiplicação* e *módulo N*, os tempos de execução são $O(n^2)$ para os ambos. Logo, o tempo resultante também será $O(n^2)$.

O método *exponenciação modular* recebe três números: a base *x*, o expoente *y* e o módulo *N*. Devolve o resultado da exponenciação em módulo *N*. Com um algoritmo recursivo, divide-se o expoente por dois até a base da recursão com *y* igual a zero. Considerando *x*, *y* e *N* com *n* dígitos cada, o algoritmo irá parar depois de aproximadamente *n* chamadas recursivas, onde são realizadas no máximo duas multiplicações modulares de tempo $O(n^2)$, resultando em um tempo de execução de $O(n^3)$.

3.4 Algoritmo de Euclides

O algoritmo de Euclides Estendido[4] encontra o máximo divisor comum de dois números inteiros dados, decide se tais números são primos relativos, e encontra o inverso multiplicativo em um número no módulo N. Este algoritmo é importante na fase da construção da chave privada.

O método *euclides estendido* recebe dois números *x* e *y*, e devolve três números: *a* o inverso multiplicativo de *x* no módulo *y*; *b* o inverso multiplicativo de *y* no módulo *x*; e, *d* o máximo divisor comum[5].

[3] Uma abordagem completa sobre o assunto pode ser vista em *"Introduction to Algorithms"* de T. H. Cormen, C. E. Leiserson, R. L. Rivest e C. Stein [Cormen et al. 2009].
[4] Mais detalhes sobre esses algoritmos encontram-se em [Dasgupta et al. 2009].
[5] Uma prova pode ser encontrada em [Cormen et al. 2009].

Figura 3. *Euclides Estendido* em pseudocódigo.

```
EUCLIDES ESTENDIDO (x, y)
1  se y igual a 0
2     então devolva (1, 0, x)
3     senão z ← x módulo y
4           (x', y', d) ← EUCLIDES ESTENDIDO (y, z)
5           devolva (y', x' − y'⌊x/y⌋, d)
```

Fonte: produzido pelos autores

Para análise de tempo são necessárias algumas provas que implicam quantos dígitos se perde a cada iteração ao realizar o módulo. A prova pode ser encontrada em [Cormen et. al. 2009]. Considerando *x* e *y* com *n* dígitos. Segundo o lema visto na literatura, *x* e *y* perdem pelo menos um dígito a cada duas chamadas recursivas onde é solicitado o método *modulo N* com tempo $O(n^2)$. Portanto, o tempo de execução é $O(n^3)$.

3.5 Números Primos

As chaves do sistema RSA são formadas a partir de número primos. Para o sistema RSA foram implementados métodos para encontrar primos relativos (ou seja, quando o máximo divisor comum de dois números é 1), realizar teste de primalidade e gerar números primos grandes. Estes algoritmos são denominados como randomizados, pois utilizam números gerados aleatoriamente. Por esse motivo as análises de tempo se tornam mais difíceis e não serão realizadas nesta seção, uma vez que, não estão relacionadas diretamente com a troca de mensagens, apenas com a geração de chaves.

O método *primo relativo* recebe um número, que se quer encontrar um primo relativo e devolve um número com o resultado. Para encontrar um primo relativo utiliza-se o método *euclides estendido*. Este método permite encontrar o máximo divisor comum entre dois números. Logo, dado um número, procura-se através da geração aleatória de outros números, um que tenha máximo divisor comum igual a 1 com o primeiro dado.

O método *teste de primalidade* recebe um número *N* e verifica se é ou não primo, retornando uma resposta positiva ou negativa. Segundo o *Pequeno Teorema de Fermat* [Dasgupta et. al. 2009], se *N* é um número primo então para qualquer número *a* inteiro positivo menor que *N*, $a^{(N-1)}$ é congruente a 1 no módulo *N*. Porém o contrário não é necessariamente verdade. O teorema não garante que somente números primos atenderão ao critério. Para diminuir as chances de o número não ser primo e passar no teste, repete-se a verificação com diferentes números *a*.

O método *gerar número primo* recebe dois números inteiros *t* e *p* e devolve um número primo com alta probabilidade. O número *t* é destinado a quantidade de dígitos desejados para o número primo e *p* é o valor destinado à certeza de o número ser primo,

ou seja, a quantidade de vezes que se realizará o teste de primalidade. Escolhe-se um número ao acaso com o tamanho *t* e realiza-se o teste de primalidade *p vezes*. Repete-se o processo até um número atender os requisitos desejados.

4 CONSIDERAÇÕES FINAIS

O Sistema de Chaves Públicas RSA atente com excelência os requisitos de segurança, facilidade de implementação e utilização. A garantia de segurança deve-se a dificuldade de descobrir a chave privada a partir da chave pública. Discussão esta envolve a fatoração do número *N*, a fim de encontrar *p* e *q*. Para longas chaves como as utilizadas pelos sistemas de criptografias RSA, a fatoração se torna um problema com alto grau de complexidade, sendo inviável realizar a fatoração das chaves públicas.

Por fim, deseja-se mostrar que a implementação de um projeto como este permite a abordagem de conceitos, presentes na ementa do curso, de forma atrativa aos alunos. Além disso, para este trabalho realizou-se o estudo sobre manipulação de números grandes, conceitos da aritmética modular, entendimento a respeito do algoritmo de Euclides e mediante a números primos, e também sobre complexidade de algoritmos.

REFERÊNCIAS

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press Cambridge.

Dasgupta, S., Papadimitriou, C., and Vazirani, U. (2009). *Algoritmos*. Mc Graw Hill.

Karatsuba, A. (1995). *The complexity of computations*. Proceedings of the Steklov Institute of Mathematics-Interperiodica Translations, 211:169-183.

Karatsuba, A. and Ofman, Y. (1963). *Multiplication of multidigit numbers on automata*. In Soviet Physycs doklady, volume 7, page 595.

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