Uma Apresentação Sucinta do Sistema de Criptografia RSA

Álvaro Junio Pereira Franco e Gabriel Estevam Oliveira

[1]Universidade Federal de Santa Catarina
Centro de Araranguá
Departamento de Computação

Abstract. *In 1978, Rivest, Shamir e Adleman developed a cryptosystem called* RSA *(first letter of their surnames). This system is used now a days, for instance, in ssh, ssl, and digital signature protocols. The main focus of this paper is to present the RSA cryptosystem in a succinct way. Initially, we describe the RSA cryptosystem. After, we discuss a way to manipulate big integer numbers, and discuss a way to implement the basic and modular arithmetic operations, the Euclid's algorithm (extended), and the generation of prime numbers. We cite the time complexity of each algorithm. Finally, we show how to use the cryptosystem RSA as a solution for the security exchange messages problem.*

Resumo. *Em 1978, Rivest, Shamir e Adleman desenvolveram um sistema de criptografia chamado* RSA *(primeiras letras dos sobrenomes dos autores). Este sistema é atualmente utilizado, por exemplo, nos protocolos ssh, ssl, e assinatura digital. O principal foco deste trabalho é apresentar o sistema de criptografia RSA de forma sucinta. Inicialmente descrevemos o sistema RSA. Depois, discutimos uma forma de manipular números inteiros grandes, e discutimos uma forma de implementar as operações aritméticas básicas e modulares, o algoritmo* estendido *de Euclides, e a geração de números primos. Citamos a complexidade de tempo de cada algoritmo. Finalmente, mostramos como utilizar o sistema RSA como solução do problema da troca segura de mensagens.*

Introdução

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, a proteção de cartões magnéticos são algumas aplicações cotidianas dos sistemas de criptografia. 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. Neste problema, duas pessoas desejam trocar mensagens com segurança, sem que um possível interceptador consiga entender o conteúdo das mesmas .

Para resolver problemas como este, podemos usar os *sistemas de criptografia com chave privada*. Neste método, é necessária a definição previamente de uma chave que irá criptografar e decriptar as mensagens. Para garantir o sigilo desta chave, ela deve ser compartilhada por outro meio, diferente do que será realizado o envio das mensagens. Para obter uma maior segurança, a chave pode ser compartilhada pessoalmente. Realizar esse encontro entre as partes pode ser muito caro e complicado.

Os *sistemas de criptografia com chave pública* é uma outra opção dada 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 (*secure shell*), ssl (*secure socket layer*), assinaturas digitais, etc. Na próxima seção, descrevemos de forma objetiva o sistema de chave pública RSA.

O Sistema de Chave Pública RSA

Primeiramente, vamos apresentar o seguinte problema clássico: Alice e Bob desejam trocar mensagens por um meio público. Eles gostariam de manter as mensagens sigilosas. Caso alguém intercepte suas mensagens no meio público, então as mensagens não poderiam ser lidas porque estariam codificadas (veja Figura 1). No sistema RSA, Alice e Bob possuem cada um, uma chave pública e uma chave privada. As chaves públicas são conhecidas por todas as pessoas. A chave privada da Alice é conhecida somente por Alice (da mesma forma ocorre para a chave privada do Bob que é conhecida somente por Bob). Se a Alice deseja enviar uma mensagens para Bob, então ela deve utilizar a chave pública do 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. Da mesma forma, para Bob enviar mensagens para Alice, ele deve utilizar a chave pública da Alice e somente ela conseguirá decriptar a mensagem enviada por Bob.

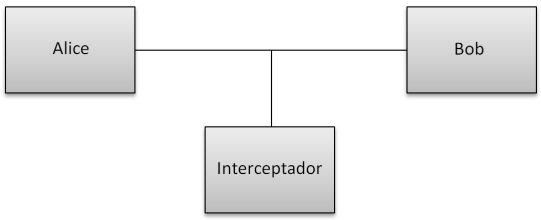

Figura 1. Cenário de troca de mensagens

Precisamos, então, formular o que são chaves públicas e privadas, e como criptografar e decriptar as mensagens. Para isso, primeiro transformamos as mensagens em números grandes, com muitos dígitos. Utilizando uma tabela de caracteres, como a UTF-8, podemos associar textos a números pareando caracteres aos códigos da tabela. Descrevemos um exemplo dessa transformação no final deste artigo (penúltima seção).

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 dos dois números primos p e q; e
- c, um *primo relativo* (veja a definição em seguida) do produto $(p-1)(q-1)$.

Dados dois números inteiros a e b, dizemos que a é *primo relativo* de b se o máximo divisor comum de a e b é 1.

Por último, a chave privada é formada por três número inteiro

- os números primos p e q usados para criar a chave pública; e
- d, o *inverso multiplicativo* no módulo $(p-1)(q-1)$ de c.

Seja X um número inteiro positivo. Sobre o *inverso multiplicativo* no módulo X de um número inteiro a, vamos nos limitar a dizer que ele é obtido através do algoritmo estendido de Euclides (descrito nas próximas seções). Existe uma vasta literatura que introduz o sistema de RSA de forma mais rigorosa. Para este intuito, sugerimos o livro "Algoritmos" de S. Dasgupta, C. Papadimitriou e U. Vazirani [Dasgupta et al. 2009].

Seja m um número inteiro não negativo no módulo N. Para criptografar m, faça

$$m' = m^c \text{ (módulo } N).$$

Ou seja, m' será o número m elevado a c-ésima potência no módulo N.

Para decriptar m', basta fazer

$$m'' = (m')^d \text{ (módulo } N).$$

Isto é, m'' será o número m' elevado a d-ésima potência no módulo N. É claro que o m original é igual ao m'' obtido. Logo, para criptografar uma mensagem basta dividi-la em blocos no módulo N e criptografar cada bloco. Para decriptar uma mensagem codificada, basta decriptar cada bloco.

Na próxima seção, descrevemos como fizemos nossa implementação deste sistema de criptografia.

Uma Implementação do Sistema RSA

O propósito desta seção é apresentar os recursos necessários para uma implementação do sistema de criptografia RSA e, consequentemente, uma solução para o problema de troca segura de mensagens. A implementação, desenvolvida em linguagem de programação Java, é código aberto, está disponível quando requisitada, e é um dos principais resultados deste trabalho.

As subseções a seguir descrevem as classes e os métodos criados. Não colocamos o código fonte da nossa implementação neste texto. Ao invés disso, descrevemos os algoritmos implementados procurando um equilíbrio entre o prazer da leitura e os detalhes de implementação. As seções que apresentam os métodos possuem somente os nomes dos métodos. Os parâmetros de cada método são destacados na própria seção.

Primeiramente, descrevemos uma proposta para o manuseio de números grandes. Em seguida, descrevemos uma forma de implementar as operações aritméticas básicas para números grandes. Da mesma forma, descrevemos as operações aritméticas modulares necessárias ao nosso propósito. Depois, apresentamos o algoritmo estendido de Euclides. Este algoritmo, além de encontrar o máximo divisor comum de dois números inteiros, ele também encontrará o inverso multiplicativo de um determinado número inteiro. Terminamos com os algoritmos que geram números primos, fundamentais para a construção das chaves pública e privada.

A Classe NÚMERO

Comentamos anteriormente que as chaves de um sistema RSA são números inteiros grandes, maiores que os tipos de dados primitivos disponíveis na linguagem. Em breve, veremos que as mensagens são tratadas também como números inteiros grandes. Por isso, para

o manuseio das chaves e mensagens, decidimos criar uma classe NÚMERO. A linguagem de programação Java possui bibliotecas que permitem trabalhar com número grandes, mas para fins didáticos[1] criamos todo o ferramental necessário para o projeto.

A classe NÚMERO possui três atributos, um construtor e um método chamado de IMPRIMIRNÚMERO. Os atributos são

- um vetor de inteiros que armazena cada dígito de um número na base 10;
- o sinal do número (positivo ou negativo); e
- a quantidade de dígitos do número.

O construtor inicializa um objeto *número* com a quantidade de dígitos igual a 0. Veja a classe NÚMERO na Figura 2.

A nossa implementação considera mais uma classe, chamada MAIN. É nela que todos os outros métodos estão implementados. Em seguida, descrevemos cada um deles.

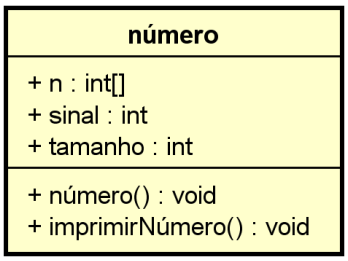

Figura 2. Diagrama de classe: NÚMERO

Aritmética Básica de Números Grandes

O sistema RSA utiliza operações básica sobre números grandes como, por exemplo, somar, subtrair e multiplicar dois números grandes, e dividir um número grande por outro. Uma nota importante, que se refere a todos os métodos, é quanto ao tamanho e o sinal dos números resultantes. Ao final das operações devemos indicar o tamanho do número resultante e estabelecer o sinal do mesmo conforme a operação realizada e a regra correspondente dos sinais.

Os Métodos SOMA e SUBTRAÇÃO

O método SOMA recebe dois números inteiros não negativos a e b (ambos com n dígitos) e devolve um número inteiro c, resultado da operação $a + b$. O algoritmo implementado, sem dificuldades, é aquele que aprendemos no ensino fundamental (ele é eficiente). Percorremos cada par de dígitos dos números a e b (da esquerda para a direita), adicionamos os dígitos deste par, e armazena o resultado da adição na posição corresponde de c. Caso a adição do par seja maior que o valor da base numérica (a base que utilizamos foi a base 10) é necessário guardar o "dígito extra" para adicioná-lo ao próximo par de dígitos. O tempo de execução desta operação é $O(n)$.

O método SUBTRAÇÃO recebe dois números inteiros não negativos a e b (ambos com n dígitos) e devolve um número c, resultado da operação $a - b$. Diferentemente da

[1]Este artigo é fruto de um projeto desenvolvido na disciplina "Projeto e Análise de Algoritmos".

soma de dois números, a ordem dos números nesta operação é importante para o sinal do resultado. Por isso, usamos um método auxiliar que determina qual dos números a e b é maior em valor absoluto. Assim, é mais facilmente realizável a diferença entre eles, subtraindo o menor do maior. Mais uma vez, não tivemos dificuldades na implementação do algoritmo básico de subtração. Percorremos cada par de dígitos de a e b, realizamos a subtração entre os dígitos do par. Se o resultado para um par for menor que zero então a subtração deste par é realizada novamente, porém, com o dígito do maior número somado ao valor da base numérica. Com isso, devemos diminuir em um o valor do próximo algarismo de c. No final do processo, o sinal de c será igual ao sinal do maior número em valor absoluto (a ou b em valor absoluto). O tempo de execução desta operação é da mesma ordem que o tempo da operação soma, isto é, $O(n)$.

Se desejamos realizar a soma ou subtração de dois números inteiros quaisquer a e b (podendo pelo menos um deles ser negativo), sugerimos ao leitor que use os métodos SOMA e SUBTRAÇÃO de tal forma que um auxilie o outro conforme o sinal dos números a e b. Por exemplo, se queremos somar a e b com $a \geq 0$ e $b < 0$, então, na verdade, realizamos a subtração $a - |b|$. Se queremos subtrair $a - b$ com $a < 0$ e $b \geq 0$, então realizamos a soma $|a| + b$, e o resultado será um número negativo. Os outros casos deixamos para o leitor tratar.

O Método MULTIPLICAÇÃO

O método MULTIPLICAÇÃO recebe dois números inteiros a e b com n dígitos cada e devolve um número c, resultado da operação ab. Mais uma vez, implementamos o algoritmo básico de multiplicação que é descrito em seguida. Para cada dígito de b (b_i), começando do último dígito até o primeiro, realizamos a multiplicação de b_i com cada dígito de a, adicionamos este resultado ao *resultado parcial* das multiplicações anteriores (inicialmente nulo). No final deste processo teremos o *resultado final* da operação (veja Figura 3). Lembramos que as somas devem ser deslocadas a quantidade de dígitos correspondentes ao dígito de b em questão. A multiplicação de um dígito de b com a pode ser feita percorrendo os dígitos de a e realizando o produto entre cada par. Caso o resultado de um produto seja maior que o valor da base numérica, devemos guardar o segundo dígito para somar ao próximo dígito do número resultante. No final do processo, a quantidade de dígitos de c será no máximo $2n$. O tempo de execução deste algoritmo é $O(n^2)$.

Implementamos o algoritmo básico para multiplicar dois números. No entanto, podemos citar o algoritmo de Karatsuba [Karatsuba 1995] e [Karatsuba and Ofman 1963] que realiza essa operação em tempo $O(n^{\log_2 3}) \approx O(n^{1,585})$, onde n é o número de dígitos dos números inteiros da entrada.

O Método DIVISÃO

O método DIVISÃO recebe dois números inteiros positivos a, com no máximo n dígitos, e b (≥ 10) com n dígitos, e devolve dois números inteiros q e r, onde q é a parte inteira da divisão de a por b, e r é o resto. O algoritmo apresentado em seguida foi implementado no nosso sistema RSA. Ele pode ser encontrado, em um outro formato, em

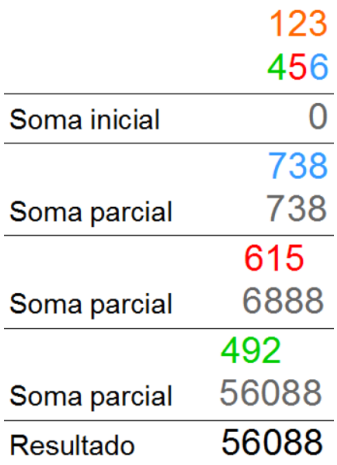

Figura 3. Um exemplo da operação de multiplicação

[Dasgupta et al. 2009].

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)
```

Note que existem n chamadas recursivas, cada uma delas realizadas depois de um deslocamento à direita em a (linha 4 do algoritmo anterior). Note também que cada chamada recursiva consome um tempo da ordem de $O(n)$ pois são realizadas as operações de deslocamento (de dígitos), adição, subtração e multiplicação de um dígito por um número inteiro. Portanto, o tempo de execução deste algoritmo é $O(n^2)$.

Aritmética Modular

Como vimos em seções anteriores, o sistema de criptografia RSA usa operações aritméticas modulares. Portanto, foram implementadas algumas operações desta aritmética.

Seja N um número inteiro positivo. A aritmética modular agrupa os número inteiros em N classes. Números inteiros pertencentes a uma mesma classe são chamados de *congruentes*. Sendo x um número inteiro, podemos dividir x por N, tendo assim um quociente q e um resto r. O número inteiro x pode ser escrito $x = qN + r$, onde $0 \leq r < N$. Logo, x módulo N é r. Uma abordagem completa sobre o assunto pode ser obtida em

"*Introduction to Algorithms*" de T. H. Cormen, C. E. Leiserson, R. L. Rivest e C. Stein [Cormen et al. 2009].

O Método MÓDULON

O método MÓDULON recebe dois números inteiros positivos x e N, ambos com n dígitos, e devolve x módulo N. Para tal tarefa, utilizamos o método DIVISÃO. Dividimos x por N. O resto da divisão, r, é x módulo N. Logo, a análise de tempo de execução para este método é a mesma para o método DIVISÃO, ou seja, $O(n^2)$.

O Método MULTIPLICAÇÃOMODULAR

O método MULTIPLICAÇÃOMODULAR recebe dois números inteiros a e b com n dígitos, e recebe também um número inteiro N correspondente ao módulo desejado. O método devolve ab módulo N. Obtemos o resultado simplesmente realizando a multiplicação entre a e b e após isso, aplicando módulo N ao resultado. O tempo de execução desta operação é $O(n^2)$, pois, os métodos MULTIPLICAÇÃO e MÓDULON possuem tempos de execução $O(n^2)$ para ambos.

O Método EXPONENCIAÇÃOMODULAR

O método EXPONENCIAÇÃOMODULAR recebe três números inteiros não negativos x, y, e N, todos com n dígitos. O método devolve o resultado da exponenciação x^y módulo N. Vamos para a descrição do método. Se $y = 0$, então devolva 1. Caso contrário, guarde o último dígito de y em u; remova o último dígito de y (colocando de outra forma, estamos realizando a divisão inteira $y \leftarrow y/10$); chame o algoritmo recursivamente com os parâmetros x, y e N, guarde o resultado em z (isto é, $z \leftarrow x^{\lfloor \frac{y}{10} \rfloor}$ módulo N); multiplique (módulo N) z por ele mesmo 10 vezes (estamos calculando $(x^{\lfloor \frac{y}{10} \rfloor})^{10}$ módulo N); multiplique (módulo N) o z atual por x^u (estamos calculando $(x^{\lfloor \frac{y}{10} \rfloor})^{10} x^u$ módulo N, obtendo assim x elevado ao y original); e finalmente, devolva z.

Note que o número de chamadas recursivas deste algoritmo é $O(n)$. Em cada chamada são realizadas um número constante de multiplicações modulares, cada uma consumindo tempo $O(n^2)$. Com isso, o tempo de execução deste algoritmo é $O(n^3)$.

O Algoritmo de Euclides

O famoso algoritmo de Euclides para encontrar o máximo divisor comum de dois números inteiros é também de grande importância no sistema RSA. Veremos a versão estendida deste algoritmo. Com essa versão, seremos capazes de encontrar primos relativos entre dois números inteiros (necessário para construir uma chave pública), e o inverso multiplicativo de um número inteiro em um determinado módulo (necessário para construir uma chave privada).

O Método EUCLIDESESTENDIDO

O método EUCLIDESESTENDIDO recebe dois números inteiros x e N com n dígitos, e devolve três números inteiros a, b e d, onde d é o máximo divisor comum de x e N, e $ax + bN = d$.

Considere dois números inteiros x e N ($N \geq 0$). Primeiramente, é importante dizer que se d é o máximo divisor comum de x e N, então existem *únicos* números inteiros a e b tal que $d = ax + bN$. O número x possui um inverso multiplicativo no módulo N se e somente se o máximo divisor comum de x e N é 1. Se este é o caso, então o algoritmo estendido de Euclides com parâmetros x e N, devolve 1, a e b, onde $1 = ax + bN$. O inverso multiplicativo no módulo N de x é a (no módulo N). Para os leitores interessados em mais informações indicamos novamente a consulta de [Cormen et al. 2009] e [Dasgupta et al. 2009]. Apresentamos tal algoritmo em seguida.

EUCLIDESESTENDIDO (x, N)
1 **se** $N = 0$
2 **então** devolva $(1, 0, x)$
3 **senão** $z \leftarrow x$ módulo N
4 $(x', N', d) \leftarrow$ EUCLIDESESTENDIDO (N, z)
5 devolva $(N', x' - N' \lfloor x/N \rfloor, d)$

Para uma análise do tempo de execução deste algoritmo, é necessário alguns resultados que implicam em quantos dígitos um número perde a cada iteração quando a operação módulo é realizada. Tais resultados podem ser encontrados em [Dasgupta et al. 2009] e [Cormen et al. 2009]. Um importante lema encontrado na literatura garante que se $x \geq N$, então x módulo N é menor que $x/2$. Isso significa que x e N perdem pelo menos um bit a cada duas chamadas recursivas consecutivas. Logo, a cada oito chamadas recursivas consecutivas, x e N perdem pelo menos um dígito (depois de oito chamadas recursivas x módulo N é menor que $x/16$). Como em cada chamada são solicitados os métodos MÓDULON e DIVISÃO com tempo $O(n^2)$ cada, o tempo de execução do algoritmo estendido de Euclides é $O(n^3)$.

Números Primos

Como visto na apresentação do problema, as chaves do sistema RSA são formadas a partir de número primos. Um número é considerado primo caso for divisível apenas por um e por ele mesmo. No nosso sistema RSA foram implementados métodos para encontrar primos relativos, para realizar testes de primalidade, e para gerar números primos. O último método implementa um algoritmo probabilístico. Começamos descrevendo o método que encontra primos relativos.

O Método PRIMORELATIVO

O método PRIMORELATIVO recebe um número inteiro x, e devolve c tal que o máximo divisor comum de x e c é 1 (ou seja, x e c são primos relativos).

É importante lembrar que este método é usado para encontrar um dos números inteiros da chave pública do sistema RSA (veja novamente a seção seguinte à Introdução).

Note que é também importante c ser tão pequeno quanto possível, para termos maior rapidez no processo de criptografia da mensagem. Utilizamos o método EUCLIDESESTENDIDO para encontrar um primo relativo de um número x. Associamos a c um valor pequeno $(2, 3, 4, 5, \ldots)$ e verificamos se o máximo divisor comum de x e c é 1. O processo continua até encontrar tal c.

O Método TESTEPRIMALIDADE

O método TESTEPRIMALIDADE recebe dois número inteiros positivos N e a (menor que N), ambos com n dígitos, e devolve uma resposta positiva se N passa no *teste do Pequeno Teorema de Fermat*, ou uma resposta negativa caso contrário.

O *Pequeno Teorema de Fermat* [Hefez 2006] está enunciado em seguida. *Se N é um número primo, então para qualquer número inteiro positivo x menor que N, x^{N-1} no módulo N é igual a 1*. Note que a contrapositiva deste teorema nos diz o seguinte. Se existir algum x menor que N com x^{N-1} no módulo N diferente de 1, então N é composto. Ou seja, podemos verificar se um dado N é composto calculando a^{N-1} módulo N e depois checando se o resultado é diferente de 1. Calcular a^{N-1} módulo N consome um tempo de execução igual a $O(n^3)$. Logo, o tempo de execução deste método é $O(n^3)$.

Supondo um N composto, para diminuir as chances deste método dar uma resposta errada, ou seja, o método responder que N é primo, devemos repetir o teste com diferentes números a $(< N)$ escolhidos ao acaso. Mas isso é tarefa do próximo método.

O Método GERARNÚMEROPRIMO

O método GERARNÚMEROPRIMO recebe dois números inteiros n e k, e devolve, com alta probabilidade (igual $1 - \frac{1}{2^k}$), um número primo P com n dígitos.

Para este método, escolhemos ao acaso cada um dos n dígitos de P e realizamos k testes de primalidade sobre P. Se o teste falha para P, então escolhemos ao acaso outro número. O método para quando P passa no teste. Note que para $k = 7$, o método devolve um número primo com probabilidade aproximadamente igual a 0.9922.

Quantas vezes devemos gerar P até que este método pare? De acordo com o *Teorema de Langrange*, dado um número inteiro positivo x, a quantidade de números primos menores ou iguais a x é aproximadamente $x/\ln x$, ou seja, existem muitos números primos menores ou iguais a x. Desta forma, se em cada iteração gerarmos um P ao acaso de maneira aleatória e independente, então a probabilidade deste método parar é aproximadamente $10^n/((\ln 10^n)10^n) = 1/\ln 10^n \approx 1/n$. Portanto, a probabilidade do método **não** parar durante, digamos, $5n$ iterações é aproximadamente $(1 - 1/n)^{5n}$. Logo, a probabilidade do método parar durante $5n$ iterações é aproximadamente $1 - (1 - 1/n)^{5n}$. Se $n = 100$, então a probabilidade do método parar nessas condições é aproximadamente 0.99343. Portanto, com alta probabilidade, o método para depois de $O(n)$ iterações. Como em cada iteração realizamos, digamos $k = 7$ testes de Fermat, o tempo de execução deste algoritmo é $O(n^4)$. Indicamos mais uma vez o livro [Dasgupta et al. 2009] para mais informações.

Criptografando e Decriptando uma Mensagem

Como uma demonstração do funcionamento do sistema RSA, a mensagem

Olá mundo!

será criptografada e decriptada. Vamos usar a tabela de caracteres UTF-8 (em decimal) para obter a mensagem original como uma sequência numérica: 079 108 225 032 109 117 110 100 111 033. Precisamos definir as chaves para o sistema RSA. Escolhidos dois números primos, $p = 41$ e $q = 23$ (elementos da chave secreta), podemos calcular o produto entre eles e um primo relativo ao produto $(p-1)(q-1)$. Assim, encontramos a seguinte chave pública: $N = 943$ e $c = 3$. Agora vamos encontrar d, o último elemento da chave privada. Para isso, basta calcular o inverso multiplicativo no módulo $(p-1)(q-1)$ de c: $d = 587$.

Podemos agora criptografar bloco a bloco da mensagem. Cada bloco m deve estar no módulo $N = 943$. Note que isso já acontece para o nosso caso. Para cada bloco m, encontramos $m' = m^c$ (módulo N) como definido anteriormente. Dessa forma, a mensagem original é transformada em 793 807 128 706 290 399 427 420 281 103. Observe que alguns blocos da mensagem criptografada não são compreensíveis na tabela UTF-8.

Para decriptar a mensagem, encontramos $m'' = (m')^d$ (módulo N) para cada bloco m'. É claro que obtemos como resultado a mensagem numérica original: 079 108 225 032 109 117 110 100 111 033.

Realizando o procedimento reverso com a tabela UTF-8, obtemos novamente a mensagem: *Olá mundo!*

Considerações Finais

Métodos de criptografia são de grande importância para a transmissão de conteúdo com garantia de proteção de dados. Em particular, o sistema de criptografia RSA atende com excelência os requisitos de segurança e facilidade de implementação e utilização. O problema da troca de mensagens visto neste artigo é apenas um exemplo das muitas aplicações possíveis para este sistema.

A garantia de segurança é obtida devido à 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. Chaves privadas pequenas, como vimos na demonstração (seção anterior), podem ser facilmente encontradas. Porém, para chaves longas, como as utilizadas pelos esquemas de criptografias reais (com aproximadamente 300 dígitos decimais), a fatoração se torna um problema com alto grau de complexidade.

Por fim, a principal finalidade deste trabalho se fez em apresentar uma implementação simples de um sistema de criptografia eficiente. Especificamente para esta tarefa, foi necessário o aprendizado sobre manipulação de números grandes, conceitos da aritmética modular, entendimento a respeito do algoritmo de Euclides e também mediante a números primos.

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.

Hefez, A. (2006). *Elementos de aritmética*. Sociedade Brasileira de Matemática.

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

Karatsuba, A. and Ofman, Y. (1963). Multiplication of multidigit numbers on automata. In *Soviet physics 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.