REVISTA DE **EMPREENDEDORISMO, INOVAÇÃO E TECNOLOGIA**

Uma análise da complexidade do algoritmo RSA implementado com o teste probabilístico de Miller-Rabin

Complexity analysis of RSA algorithm implementation using Miller-Rabin probabilistic test

Ricardo de la Rocha Ladeira(1); Anderson Schwede Raugust(2)

1 Instituto Federal Catarinense (IFC) - Campus Blumenau – Brasil. E-mail: ricardo.ladeira@ifc.edu.br
2 Universidade do Estado de Santa Catarina (UDESC) – Brasil. E-mail: anderson.raug@gmail.com

Revista de Empreendedorismo, Inovação e Tecnologia, Passo Fundo, vol. 4, n. 1 p. 24-33, Jan.-Jun. 2017 - ISSN 2359-3539

DOI: https://doi.org/10.18256/2359-3539/reit-imed.v4n1p24-33

Endereço correspondente / Correspondence address
Ricardo de la Rocha Ladeira
Rua Bernardino José de Oliveira, nº 81 - Blumenau – SC.
CEP: 89070-270

Como citar este artigo / How to cite item: clique aqui!/click here!

Resumo

A criptografia de chave pública foi um marco na evolução da comunicação, tornando possível garantir uma comunicação segura em um ambiente interconectado aberto, como a internet. Este artigo descreve o funcionamento da criptografia de chaves públicas, com ênfase no algoritmo RSA. Apresenta-se o funcionamento do algoritmo de forma matemática, bem como sua aplicabilidade. Estão descritas as estratégias de implementação do RSA com o uso do algoritmo probabilístico de Miller-Rabin e a complexidade de funções construídas. Através de uma implementação na linguagem Java, foi possível gerar chaves de tamanho parametrizado em termos de *bits* e testar a sua segurança, utilizando um algoritmo de força bruta para a quebra, além de analisar os tempos de execução para as funções de geração e quebra de chaves. Foi possível observar, através dos resultados, que as chaves foram geradas em tempo polinomial, enquanto a quebra necessitou de tempo exponencial ao tamanho da chave, dada em *bits*. Portanto, a quebra de uma chave de grande tamanho através de força bruta mostrou-se impraticável.

Palavras-chave: Criptografia, Chave pública, RSA

Abstract

The public key cryptography was a mark in the evolution of the communication, making possible to ensure the security of a communication over an open interconnected environment, such as the internet. This paper describes the behavior of the popular public key cryptography, focusing on RSA algorithm. It shows how the algorithm works in a mathematical form as well as its applicability. The RSA's implementation strategies are described with Miller-Rabin probability algorithm and complexity of built functions. With an implementation in Java language, it was possible to generate parameterized keys size in terms of bits and test their security. For the security test, a brute force algorithm was used for the key break. With the results, was possible to observe that the keys were generated in polynomial time, whereas the break required exponential time to the key size, given in bits. Thus, breaking a large key through brute force has shown impractical.

Keywords: Cryptography, Public Key, RSA

Criptografia *assimétrica* ou *de chave pública* é uma forma de tornar mensagens ilegíveis utilizando um par de chaves, tipicamente chamadas *chave pública* e *chave privada*. Esta abordagem traz uma vantagem em relação à criptografia simétrica: a facilidade no gerenciamento de chaves. Segundo Kurose e Ross (2006), uma dificuldade da criptografia simétrica é a necessidade de emissor e receptor concordarem, de alguma maneira, com a chave compartilhada. Mas para fazê-lo é necessário um canal de comunicação seguro.

O algoritmo que implementa a criptografia assimétrica, ao gerar um par de chaves, garante matematicamente que a mensagem cifrada com uma das chaves somente poderá ser decifrada pela outra chave. Uma analogia pode ser feita com um cofre: suponha que exista uma chave (pública) para trancá-lo, mas ela não consegue abrir; somente a outra chave (privada) é capaz de abri-lo. A Figura 1 apresenta um exemplo de uso da criptografia assimétrica:

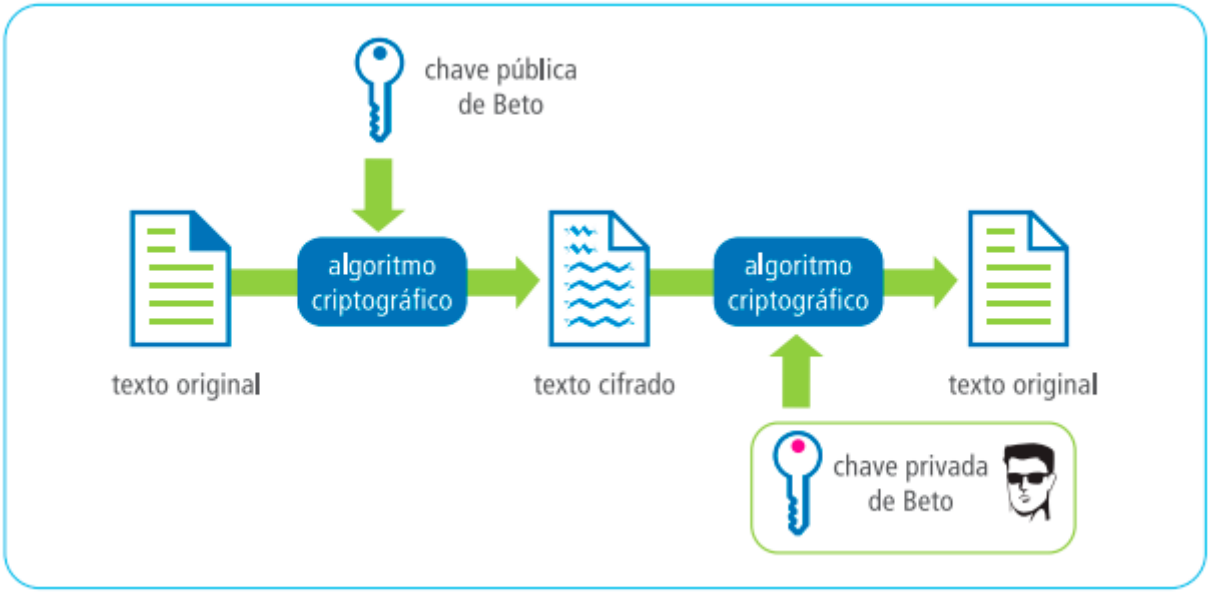

Figura 1. Sigilo utilizando criptografia assimétrica.
Fonte: ITI, 2005.

Como visto na Figura 1, o texto cifrado pela chave pública de Beto é decifrado pela chave privada, sendo possível somente assim o retorno à mensagem original.

No algoritmo RSA, grande parte da segurança está relacionada ao desconhecimento de uma função que fatore números primos em tempo polinomial. O teste de primalidade de Miller-Rabin pode ajudar nesta tarefa ao indicar quando um número é *provavelmente* primo. Este teste é comumente utilizado para ataques ao RSA (Kantarcioglu, 2007). Neste trabalho o algoritmo de Miller-Rabin está aplicado na geração das chaves.

Atualmente, protocolos seguros executados na Internet utilizam algoritmos de criptografia assimétrica, entre os quais encontram-se Diffie-Hellman (Diffie & Hellman, 1976) e RSA (Rivest, Shamir, & Adleman, 1978). Estes algoritmos são utilizados para geração de chaves, geralmente associadas a um certificado digital, e

encriptação de mensagens. A Infraestrutura de Chaves Públicas do Brasil – ICP-Brasil recomenda o uso do algoritmo RSA como padrão para geração de chaves criptográficas de, no mínimo, 2048 *bits* (ICP-Brasil, 2009).

Duas propriedades de Segurança da Informação são atendidas com base na forma com que as chaves são utilizadas no processo de cifragem, considerando que a chave privada não está comprometida. Segundo o Instituto Nacional de Tecnologia da Informação (ITI, 2005), pode-se satisfazer as propriedades de *confidencialidade* (1) ou de *autenticidade* (2) das informações quando um algoritmo de criptografia assimétrica é utilizado. Para o caso (1), utiliza-se a chave pública do remetente no processo de cifrar a mensagem. Assim, somente o próprio remetente poderá decifrá-la, garantindo o seu sigilo. No caso (2), o emissor da mensagem pode cifrá-la com a sua chave privada; desta forma, qualquer pessoa que tiver acesso à mensagem poderá, através da chave pública do emissor, confirmar que a mensagem é da entidade emissora, pois esta seria a única capaz de enviar uma mensagem com a chave privada.

Neste trabalho, pretende-se apresentar o algoritmo RSA, bem como uma análise de complexidade de sua implementação com o teste probabilístico de Miller-Rabin. O trabalho está organizado em quatro capítulos, incluindo esta introdução. O capítulo 2 versa sobre o algoritmo RSA, explicando seus conceitos e funcionamento. O capítulo 3 aborda a implementação do algoritmo, contendo metodologia, complexidade e análise das funções desenvolvidas, e o capítulo 4 contém as considerações finais.

RSA

O RSA, cujo nome advém das iniciais dos sobrenomes dos autores, foi apresentado em um artigo publicado, em 1978, por Ronald Rivest, Adi Shamir e Leonard Adleman. Nessa publicação, descreveu-se o algoritmo como sendo um método de criptografia com uma nova propriedade: revelar publicamente uma chave de criptografia e esconder a chave de decodificação correspondente, sendo apresentadas suas vantagens, definição teórica e propriedades atingidas (Rivest et al., 1978).

Na sequência da seção discorre-se sobre o funcionamento do algoritmo RSA.

Funcionamento

O RSA está fortemente ligado à Teoria dos Números, sendo baseado em pilares como as operações de resto e fatoração por números primos. O algoritmo pode ser resumido nos passos descritos abaixo (Rivest et at., 1978):

1. Obter dois números primos p e q;
2. Calcular $n = pq$;
3. Calcular $\Phi(n) = (p-1)(q-1)$;

4. Escolher e | Máximo Divisor Comum – MDC entre e e $\Phi(n)$ seja igual a 1, ou seja, e e $\Phi(n)$ são coprimos (primos relativos);
5. Calcular d | $de \equiv 1 \ (mod \ \Phi(n))$, ou seja, $de \ mod \ \Phi(n) = 1$;
6. Chave pública: (e, n); chave privada: (d, n);
7. Função para cifrar uma mensagem m: $C(m) = m^e \ mod \ n = c$;
8. Função para decifrar uma mensagem c: $D(c) = c^d \ mod \ n = m$;
9. $D(C(m)) = m$.

Cabe ressaltar que os valores $\{d,e\}$ estão unidos matematicamente. Isto pode ser percebido através de uma propriedade denominada inverso multiplicativo, ou seja, multiplicando o resultado de $d \ mod \ \Phi(n)$ pelo resultado de $e \ mod \ \Phi(n)$ obtém-se $\Phi(n) + 1$. Assim, sabe-se que $\Phi(n) + 1 \ mod \ \Phi(n) = 1$, o que garante a propriedade.

O RSA se mantém devido à dificuldade em fatorar um grande número (n) em números primos (p e q). Se b é o número de *bits* de n, então existem $\sqrt{(2^b-1)}$ possibilidades a serem testadas em um eventual pior caso, o que resulta em complexidade de tempo de $O(\sqrt{(2^b)})$. A título de curiosidade, considerando $b = 2048$, $\sqrt{(2^b)}$ resulta em um número um pouco maior que $1{,}79.10^{308}$. Considerando uma supermáquina que consegue processar 1 bilhão (10^9) de tentativas por segundo, seriam necessários mais de 5.10^{291} anos.

Implementação

O algoritmo RSA foi implementado na linguagem Java, a fim de observar na prática o seu funcionamento. O trabalho consistiu na implementação da criptografia e da descriptografia de um arquivo de texto usando o RSA e um algoritmo de força bruta para quebra da chave criptográfica.

Na sequência da seção, apresenta-se a metodologia utilizada, a complexidade das funções implementadas e análise do tempo de execução da geração do par de chaves e da fatoração de inteiros de *n bits*.

Método

Para implementar o RSA, o primeiro passo realizado foi a geração de dois números primos aleatórios, do tipo *BigInteger*. A primalidade foi garantida utilizando um algoritmo probabilístico chamado Miller-Rabin (Conrad, s.d.). Este algoritmo era responsável por informar que *provavelmente* os números gerados poderiam ser primos e, portanto, atribuídos a p e q. A atribuição do número primo à variável q foi inserido em um *loop do.. While*, para impedir que os valores de p e q sejam iguais e, com isso, gerar arquivos cifrados iguais aos arquivos em texto claro. Depois, foi necessário multiplicar os valores p e q e armazenar o resultado em uma variável n também de tipo *BigInteger*. O valor de $\Phi(n)$ foi calculado decrementando em 1 unidade os valores de p

e *q* e passando diretamente a multiplicação destes valores para a função que calcula o inverso multiplicativo de *e*, através do algoritmo de Euclides Estendido.

A geração do par de chaves se deu, inicialmente, encontrando um valor para *e*, que precisaria ser coprimo de *Φ(n)*. Utilizou-se a estratégia de fixar o tamanho de *e* em 15 *bits* para conferir segurança à medida que o tamanho da chave cresce. Desta forma o valor de *d*, contido na chave privada, é maior, dificultando que este seja descoberto por força bruta. Através do algoritmo de Euclides Estendido foi obtido o valor de *d*, pois este algoritmo implementa o inverso multiplicativo (Knuth, 1997). Isso significa que, dado *e*, ele descobre um valor para *d* que satisfaz o item 5 da subseção "Funcionamento" (*de ≡ 1 (mod Φ(n))*, ou seja, *de mod Φ(n) = 1*). Deste modo, *(e,n)* e *(d,n)*, o que significa dizer que as chaves pública e privada, respectivamente, estão geradas.

A função para criptografar foi feita obtendo *c*, o resto da divisão entre m^e e *n*, onde *(e, n)* formam a chave pública e *m* é a mensagem em texto claro. A função de descriptografar foi feita a partir da mensagem *c*, já criptografada, e da chave privada *(d, n)*, calculando o resto da divisão entre c^e e *n*. Para isso, foi necessário apenas converter os caracteres do arquivo de texto em um valor inteiro, que seria codificado com o algoritmo RSA, e utilizar o novo valor inteiro gerado para fazer o processo inverso: transformá-lo em caractere e escrevê-lo em um arquivo novo cifrado.

Após realizar testes das funções de cifragem e decifragem utilizando o algoritmo RSA construído, implementou-se também um algoritmo de força bruta para a fatoração da chave pública nos números primos que a geraram. A estratégia utilizada foi de utilizar um laço de repetição gerando um número iterador *i* (também *BigInteger*) que inicia em *√n* e decresce até *2*. Quando este número é divisível por *n*, ou seja, quando *i mod n = 0*, o programa encontra *p*. Pode-se dizer que também encontra *q*, pois tendo *q* este valor é obtido em tempo constante através da divisão *n/p*. Um comando *break* foi adicionado para que o algoritmo não percorra todo laço após encontrar um valor. Este algoritmo foi repetido para *n* contendo *16*, *24*, *32*, *40*, *48*, *56*, *64* e *72 bits*. Para obter o valor de *d*, calculou-se o valor de *Φ(n)* e utilizou-se o já citado algoritmo de Euclides Estendido.

Complexidade das funções implementadas

A geração do par de chaves foi feita utilizando o algoritmo Miller-Rabin. Este algoritmo foi executado duas vezes, uma para gerar o número primo *p* e outra para o número primo *q*. A complexidade do algoritmo é *O(k log2 n)*, sendo *k* a precisão do algoritmo, ou número de iterações. Para obter os inversos multiplicativos *d* e *e*, utilizou-se o algoritmo de Euclides Estendido. Lembrando que *d* e *e* precisavam satisfazer a condição *de ≡ 1 (mod Φ(n))*. Neste algoritmo, implementou-se uma função para obter o MDC de forma recursiva. A complexidade do algoritmo de Euclides Estendido é $O(\log^2 \Phi(n))$. Em termos de *bits*, pode-se dizer que o algoritmo

apresenta complexidade polinomial. De uma forma geral, a geração de chaves pode ser considerada uma atividade executada em tempo polinomial.

A fatoração em números primos para um inteiro *m* de *n bits* foi implementada com força bruta, verificando, valor por valor, qual pode ser um divisor de *m*. O iterador inicia em $\sqrt{m}$ e termina em *2* no laço de repetição, portanto os comandos internos do laço são executados, no máximo, $(\sqrt{m})-1$ vezes, o que resulta em complexidade $O(\sqrt{m})$. Em termos de *bits*, pode-se dizer que um número *m* utiliza $b = log\ m$ *bits*, sendo a complexidade $O(\sqrt{2^b}) = O(2^{b/2}) = O(2^b)$ em termos do número de *bits*. No laço de repetição há ainda operações polinomiais em termos de *bits*, que são *mod* e multiplicação. Pode-se dizer, portanto, que para quebrar a chave é demandado tempo exponencial.

Análise da execução da geração de chaves e fatoração para inteiros com *n bits*

Abaixo estão apresentados os gráficos da execução da geração das chaves e da fatoração para inteiros com *n bits*. A Figura 1 contém o gráfico com o tempo de execução necessário para gerar o par de chaves com 512, 1024, 1536, 2048, 2560, 3072, 3584 e 4096 *bits*. Para cada tamanho de chave, o algoritmo foi executado 10 vezes e a média aritmética simples foi calculada. Com base na Figura 2, é possível perceber que o crescimento da chave aumenta consideravelmente o tempo de execução do algoritmo.

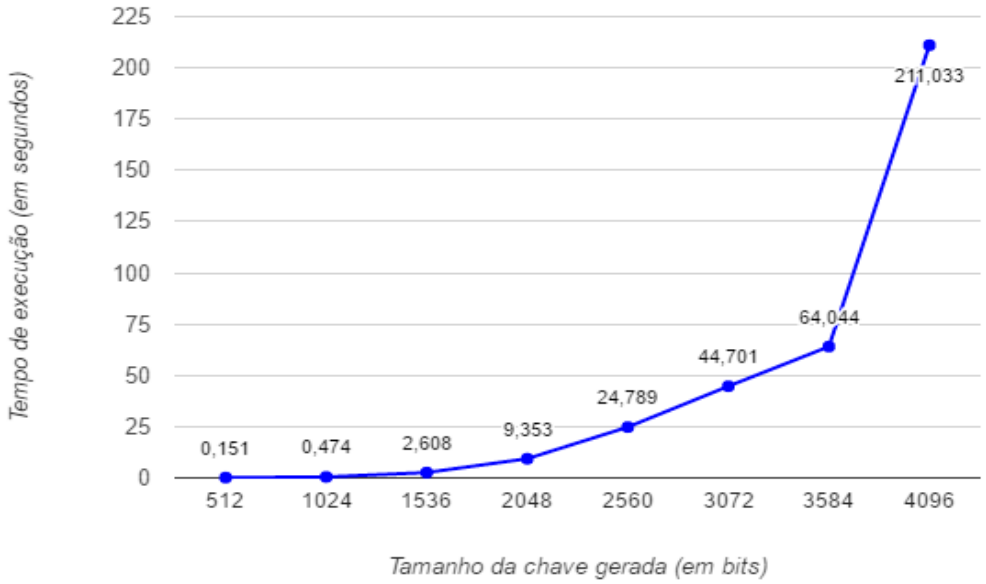

Figura 2. Média de tempo de execução do algoritmo de geração do par de chaves em função do tamanho das chaves.
Fonte: elaborado pelos autores.

O algoritmo de força bruta que calcula a fatoração em números primos para inteiros com *n bits* tem sua execução representada na Figura 3, exposta a seguir. Foram

testados valores com 16, 24, 32, 40, 48, 56, 64 e 72 *bits*, e, para cada tamanho de chave, o algoritmo foi executado 10 vezes e a média aritmética simples foi calculada. Para fins de clareza no gráfico, o resultado da quebra de chave de 72 *bits* foi omitido. Neste caso, a média de tempo de execução foi de 2795 segundos (mais de 46 minutos).

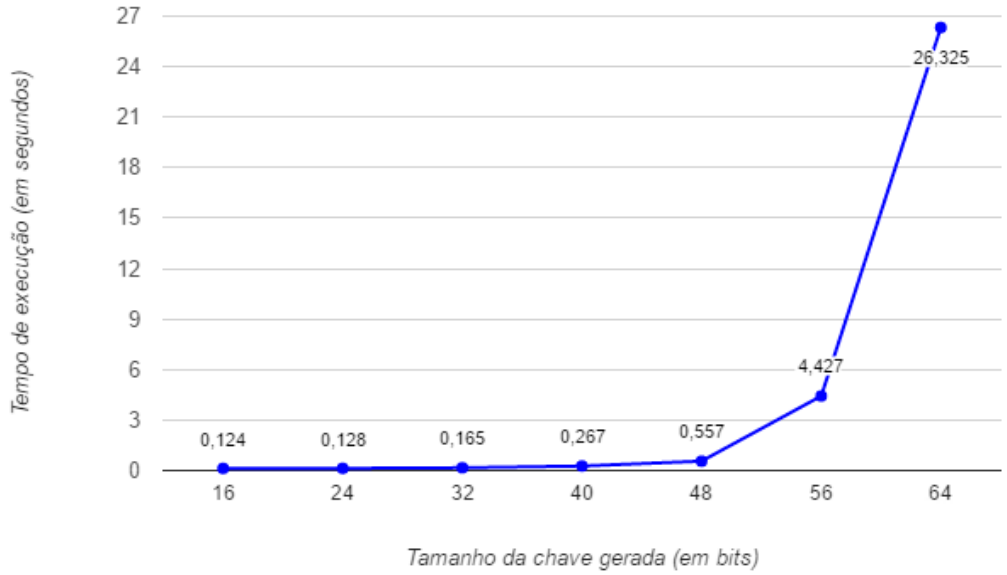

Figura 3. Média de tempo de execução do algoritmo de quebra em termos do tamanho da chave.
Fonte: elaborado pelos autores.

Com base nos resultados, é possível perceber que o tempo de execução cresceu exponencialmente, e, comparado aos tempos de geração de chaves exibidos na Figura 2, o tempo de execução para quebra é muito maior. Por exemplo, uma chave de 2048 *bits* é gerada, em média, em pouco menos de 10 segundos, enquanto uma chave de 64 *bits* é quebrada, em média, em pouco mais de 26 segundos.

Durante a escrita do artigo, tentou-se também realizar a quebra de chave para 80 *bits*, mas o valor foi descoberto em cerca de 15 horas e, por este motivo, o teste foi realizado apenas uma vez.

Uma análise mais profunda é necessária, com a realização de mais testes para cada tamanho de chave, bem como uma análise com intervalo de confiança, pois chaves de tamanho 16 e 24, por exemplo, apresentaram tempos de execução muito próximos. Além disso, entende-se que para chaves menores os custos constantes do algoritmo pesam mais que o tamanho da chave e, por este motivo, os valores são muito próximos. A medida que o tamanho da chave cresce, a diferença torna-se mais perceptível.

Considerações Finais

O RSA é um exemplo de algoritmo de criptografia assimétrica utilizado em larga escala nos dias de hoje. Ao expor o funcionamento do algoritmo, ficou clara a

dificuldade de fatorar um número inteiro grande em números primos, tarefa para qual não se conhece solução em tempo polinomial.

Com base na implementação apresentada, percebe-se que o custo de tanta segurança é o desempenho, pois é um algoritmo computacionalmente custoso. Para quebra de chaves, a complexidade do algoritmo de força bruta implementado é $O(2^b)$, sendo b o número de *bits* de n. Para a geração de chaves, onde se utilizou o algoritmo probabilístico Miller-Rabin, de complexidade $O(k \log^2 n)$, e o algoritmo de Euclides Estendido, executado em tempo polinomial. Fica claro que a geração de chaves é realizada em tempo polinomial, mas a quebra da chave exige tempo exponencial, e é isso que suporta a segurança do algoritmo RSA.

A complexidade das funções apresentadas indica que, utilizando uma chave muito grande (ex: 2048 *bits*), é impraticável, pelo menos através de algoritmo de força bruta, quebrá-la em um tempo razoável. O algoritmo implementado estava de acordo com a complexidade calculada.

Observou-se no trabalho a análise do tempo despendido na geração do par de chaves e na fatoração dos números primos que geraram n, para vários valores, demonstrando graficamente o custo da fatoração em números primos, especialmente ao aumentar o número de *bits* de n.

A continuidade do trabalho se dará seguindo a linha de comparação de desempenho em termos de tempo de execução entre diferentes implementações do RSA. Pretende-se ainda estender o estudo de algoritmos probabilísticos através da implementação de heurísticas para reduzir o tempo necessário para realizar a quebra da chave privada.

Agradecimentos

Os autores agradecem ao Instituto Federal Catarinense, por meio do Programa Institucional de Qualificação de servidores para o Instituto Federal Catarinense (PIQIFC), e à Universidade do Estado de Santa Catarina, que tornaram possível a realização deste trabalho.

Referências

Conrad, Keith. (s.d). *The Miller-Rabin Test*. Disponível em: http://www.math.uconn.edu/~kconrad/blurbs/ugradnumthy/millerrabin.pdf. Acesso em: 15 nov. 2016.

Diffie, W; Hellman, M. E. (1976). New Directions in Cryptography. *IEEE Transactíons on Information Theory*, vol. IT-22, 644-654.

ICP-BRASIL. (2009). "Plano de adoção de novos padrões criptográficos". Disponível em: http://www.iti.gov.br/images/icp-brasil/Normas%20ICP-Brasil/Plano%20de%20Adocao%20de%20novos%20padroes%20criptograficos/PLANO_DE_ADOCAO.pdf. Acesso em: 12 out. 2016.

ITI. (2005). *O que é certificação digital?* Cartilha. Disponível em: http://www.iti.gov.br/images/publicacoes/cartilhas/cartilhaentenda.pdf. Acesso em: 21 jul. 2016.

Kantarcioglu, Murat. (2007). *Primality Testing and Attacks on RSA*. Disponível em: http://www.utdallas.edu/~muratk/courses/crypto07_files/rsa-attacks.pdf. Acesso em: 15 nov. 2016.

Knuth, Donald E. (1997). *The Art of Computer Programming*, Volume 1 (3rd Ed.): Fundamental Algorithms. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Rivest, R. L., Shamir, A., & Adleman, L. M. (1978, Fev.). A Method for Obtaining Digital Signatures and Publk-Key Cryptosystems. *Communications of the ACM, 21*(2), 120-126.