# ALGORITMO DE CRIPTOGRAFIA RSA:
## análise entre a segurança e velocidade[1]


## RESUMO

O objetivo deste trabalho é abordar a relação existente entre a busca pela segurança de dados e a velocidade de codificação e decodificação do algoritmo de criptografia RSA, que utiliza um par de números inteiros como 'chave'. Considerando o tamanho da chave como requisito de segurança, devido à dificuldade computacional de fatorar números inteiros extensos, simulamos estes processos, com o algoritmo implementado na linguagem de programação C, utilizando chaves aleatórias de 1024, 2048 e 4096 *bits*. Desta forma, observamos o tempo de processamento em função do tamanho das chaves, confrontando segurança e desempenho.



**Palavras-chave**: Criptografia. Algoritmo RSA. Segurança. Desempenho.


## 1 INTRODUÇÃO

No passado, as comunicações entre os povos distantes eram feitas através de mensagens escritas. Mas temendo que elas fossem interceptadas por outras pessoas, foi desenvolvido, com o passar dos tempos, várias técnicas para tentar ocultar a mensagem através de códigos secretos ou cifras (SILVA, 2006). A base desta ideia está na cumplicidade entre emissor e receptor final da mensagem, caso contrário, decifrá-la dependeria fortemente da habilidade do receptor em decifrar.

---

[1] Artigo apresentado como Trabalho de Conclusão de Curso (TCC) do Curso de Licenciatura em Matemática com Enfoque em Informática, da Universidade Estadual do Sudoeste da Bahia (UESB), *campus* de Jequié, em 2009, sob a orientação do professor Msc.Fernando dos Santos Silva.


[*] Graduado em Licenciatura em Matemática com Enfoque em Informática, pela Universidade Estadual do Sudoeste da Bahia (UESB) - *campus* de Jequié.

[**] Mestre em Matemática pela Universidade Estadual de Campinas (Unicamp). Professor da Universidade Estadual do Sudoeste da Bahia (UESB) - *campus* de Jequié.

A criptografia trata de métodos e técnicas para transformar a mensagem em outra, de difícil compreensão, em que somente o seu destinatário legítimo possa decifrá-la, tendo assim acesso à mensagem inicial. O ato de transformar a mensagem chama-se cifrar e é um processo denominado de criptografar sendo o processo inverso, para recuperar a mensagem original, chamado de descriptografar e a mensagem resultante do processo de criptografar é chamada de mensagem cifrada.

Com o desenvolvimento do computador, os governos e as empresas passaram também a armazenar suas informações em bancos de dados. Inicialmente a criptografia foi usada na proteção dos mesmos, dificultando o acesso não autorizado (FARIA, 2006). Atualmente é de uso comum, principalmente utilizada em bancos e empresas no comércio eletrônico, em que buscam, como requisitos de segurança, a confidencialidade, integridade e autenticação.

Numa simples troca de mensagens pela internet é utilizada a criptografia para garantir o sigilo da comunicação. As transações bancárias, por exemplo, pela internet seria inviável sem a criptografia, pois qualquer pessoa mal intencionada poderia interceptar mensagens e senhas e, com isso, as informações sigilosas dos correntistas. Além disso, trocar informações e fazer negócios não teria sentido, no comércio, sem a garantia da confiabilidade do processo.

Navegadores atuais, como por exemplo: *Mozila Firefox* e *Google Chrome*, mostram se a conexão é criptografada e, em caso positivo, o tipo da criptografia empregado e/ou o certificado digital em uso, cuja intenção impede que terceiros obtenham dados de seus usuários.

> Entidades certificadoras são instituições responsáveis pela emissão de certificados digitais que identificam sites na Internet e seus respectivos proprietários. Ao assinar digitalmente os certificados que emite, a entidade certificadora relaciona a identidade do portador do certificado, e portanto da chave privada, à chave pública existente no certificado. A maioria dos navegadores exibe se a página visitada possui um certificado digital válido, caso não possua, o site provavelmente é uma fraude. (BRAGA, 2011, p. 7)

Os algoritmos atuais utilizam como parâmetro uma chave de modo que temos uma criptografia e descriptografia diferente para cada chave diferente. Quanto maior o tamanho da chave mais difícil ela será encontrada e, portanto mais segura será a mensagem criptografada. Com o aumento do seu poder de processamento, os computadores passaram a descobrir as chaves mais facilmente, exigindo assim, o aumento gradativo das mesmas. Mas de acordo com Fuzitaki (2004) esse aumento implica na diminuição do tempo de cifragem e decifragem, resultando em uma disputa entre velocidade e segurança do algoritmo.

Neste trabalho abordaremos inicialmente a criptografia e em seguida aprofundaremos sobre o algoritmo RSA, dada a sua importância histórica e cotidiana, explorando seus fundamentos matemáticos, segurança e desempenho. A seguir, através da implementação desse algoritmo na linguagem de programação C, faremos simulações utilizando mensagens e chaves aleatórias para averiguarmos velocidades de processamento da cifragem e da decifragem em função do tamanho das chaves de 1024, 2048 e 4096 *bits*.

## 2 REFERENCIAL TEÓRICO

A palavra criptografia vem do grego *kryptós*, que significa secreto, oculto ou ineteligível e *grápho*, que denota escrever, dessa forma entendemos por criptografia uma escrita secreta. Segundo Coutinho (2000), a criptografia estuda os métodos para codificar uma mensagem, transformando-a em outra, utilizando funções matemáticas, de modo que só seu destinatário legítimo consiga interpretá-la, portanto, é a arte dos códigos secretos.

Podemos codificar um texto com lápis e papel, como exemplo temos os romanos que utilizaram o conhecimento do idioma grego em segredos de estado. O código de César (*Caesar cipher*) foi muito utilizado, por Júlio César em 50 a.C.,

> As cifras de substituição também foi o método criptográfico preferido de Júlio César. Em seu livro intitulado "Guerras Gaulesas", ele conta como enviava mensagens substituindo as letras romanas por letras gregas, tornando a mensagem inteligível aos inimigos. (OLIVEIRA, 2005, p. 4)

O método consistia em permutar cada letra da mensagem original por uma letra que ocupava três posições a frente da letra inicial, como segue nas tabelas abaixo:

Tabela 1 – Cifra de substituição monoalfabéticas do Código de César

| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | E | F | G | H | I | J | K | L | M | N | O | P |
| **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **Y** | **X** | **Z** |
| Q | R | S | T | U | V | W | Y | X | Z | A | B | C |

Fonte: Criptografia Simétrica, OLIVEIRA, 2005, p.3.

Tabela 2 - Exemplo do Código de César

| **Mensagem Original** | **LICENCIATURA EM PEDAGOGIA** |
|---|---|
| *Mensagem Cifrada* | **OLFHQFLDWXUD HP SHGDJRJLD** |

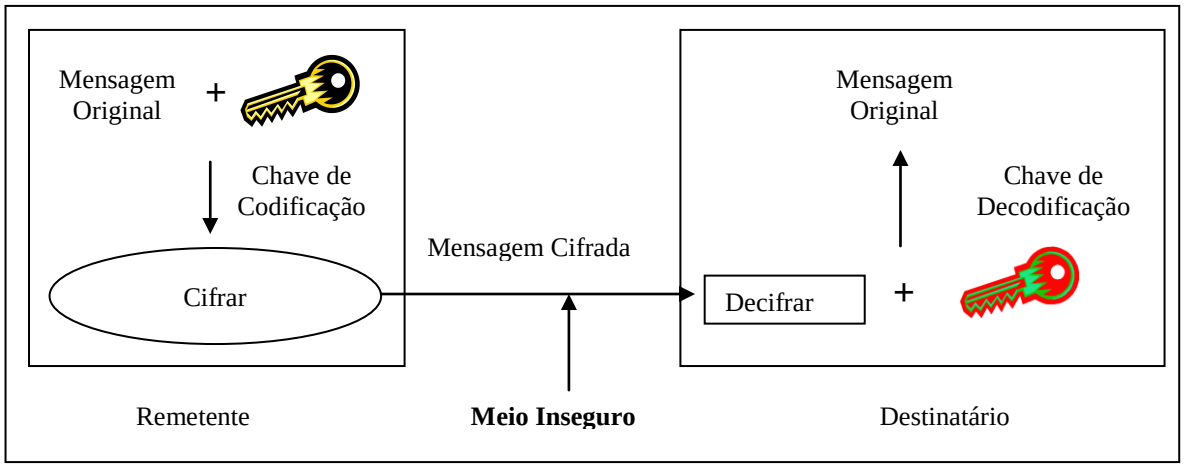
Fonte: Elaborado pelos autores, 2007.

Os tipos de codificações mais triviais, hoje em dia, são os códigos de transposição (*transposition ciphers*) que consistem em mudar a posição das letras da mensagem sem alterá-las, causando difusão em todo o texto cifrado; e os códigos de substituição (*substitution ciphers*) que sistematicamente substituem letras ou grupos de letras, produzindo confusão e relações complexas entre texto original e o cifrado.

A segurança desses métodos estava no segredo dos seus processos de cifragem. Mais tarde, os algoritmos passaram a utilizar uma chave como parâmetro, de modo que temos uma criptografia diferente para cada chave diferente. Segundo Benits Júnior (2003), ficou provado que o tamanho da chave torna a criptografia mais segura, entretanto o mesmo não ocorre com o segredo do algoritmo.

Em outras palavras, a criptografia comunica-se na presença de adversários com um esquema criptográfico seguro até mesmo se eles conhecem os algoritmos de codificação ou de decodificação. Dessa forma, o segredo da chave deve ser suficiente para a confiança do método. Podemos esquematizar os algoritmos modernos da seguinte maneira:

**Figura 1 -** Esquema geral da Criptografia




Fonte: Adaptado de Oliveira, 2005.

Com este conceito, surgiram os algoritmos mais sofisticados de criptografia como os DES, Triple-DES, IDEA, Skipjack, RC2, RC4, Lúcifer, AES , RSA, Diffie-Hellman, Merkle-Hellman, DSA e outros[2]. Com a utilização dos computadores na criptografia, as mensagens agora podem ser, além de textos, quaisquer arquivos como imagens, sons e etc. (FARIA, 2006).

[2] Para saber mais veja Fuzitaki (2004) e Barbosa et al (2003).

Existem dois tipos de criptografia: ‘simétrica e assimétrica’. Na criptografia simétrica a chave utilizada para criptografar é a mesma para descriptografar, devendo a mesma ser mantida em sigilo. Enquanto que na assimétrica existem duas chaves distintas, uma para criptografar chamada de ‘chave pública’, que pode e deve ser divulgada livremente, e uma para descriptografar, chamada de ‘chave privada’ que deve ser mantida em segredo (FUZITAKI, 2004), só deve ser conhecida apenas por um único usuário ou um grupo reservado de pessoas.

Segundo Benits Júnior (2003), um dos problemas do sistema simétrico está na transmissão segura da chave para o destinatário, porque se ela for interceptada, outra pessoa poderá decifrar a mensagem. Já a criptografia assimétrica não necessita desse canal seguro, pois, a chave privada não é transmitida. Entretanto, o sistema assimétrico é mais lento porque geralmente são baseados em cálculos extensos envolvendo problemas matemáticos ainda sem rápida solução computacionalmente, enquanto que o simétrico utiliza combinadamente as técnicas de transposição e substituição.

## 2.1 PRINCÍPIOS MATEMÁTICOS DA CRIPTOGRAFIA MODERNA

A matemática utilizada na criptografia trabalha com números inteiros e o computador com números binários. Para converter uma mensagem em uma seqüência de números usamos a *American Standard Code for Information Interchange* (**ASCII**) tabela e/ou código padrão em computadores pessoais utilizados desde a década de 60. Esta primeira etapa chama-se pré-codificação.

A tabela original trabalhava com 7 *bits*, representando 128 caracteres, inicialmente para língua inglesa que não possui caracteres acentuados, e ela foi estendida a 8 *bits* para contemplar outros idiomas. Atualmente utilizam-se **ASCII ESTENDIDA**, com o nome **ASCII**, com 256 caracteres correspondendo ao alfabeto latino, com letras maiúsculas, minúsculas, letras acentuadas, pontuação e outros símbolos.

Na criptografia o método comum é adotar o código **ASCII**, por exemplo, a letra ‘A’ é representada pela sequência $(01000001)_2$, que transformando na base 10 é igual a $65 = 0x2^7+ +1x2^6+0x2^5+0x2^4+0x2^3+0x2^2+0x2^1+1x2^0 = 64+1$. Mensagens agora são números que podem ser extremamente gigantes. Por exemplo, a simples mensagem ‘RSA’, após pré-codificação letra a letra, transforma-se em ‘828365’.

Mesmo antes do Império Romano a criptografia continua com seu princípio básico de encontrar uma função *f* bijetora entre conjunto de mensagens, escrita numa determinada lista

de símbolos, para um conjunto de mensagens codificadas com os mesmos símbolos. O receptor pode reverter à mensagem codificada, pois conhece a inversa da *f*..

O desafio contínuo da criptografia é encontrar funções de um só sentido e/ou unidirecionais que nos permitam usar a função no sentido direto com cálculo fácil para cifrar e decifrar (somente usuários legítimos), sendo que para determinar a função inversa seja difícil ou impossível para outros usuários. A cifra de César, é um contra exemplo, quando enumeramos as letras de 01 a 26, a função dada por $f(M_i) = M_i + 3 \pmod{26}$, onde $M_i$ é a letra de número i, tem fácil inversão dada por $f^{-1}(f(M_i)) = M_i - 3 \pmod{26}$.

Somente com a matemática moderna e o avanço dos computadores foi possível definir boas funções unidirecionais. A escolha da função unidirecional pode variar com o tempo, por exemplo, um cálculo requerendo $10^6$ operações e 10K palavras de memória, há 40 anos seria muito difícil, porém para um computador portátil é trivial.

## 2.2 ALGORITMO DE CRIPTOGRAFIA RSA

Dentre os vários algoritmos de criptografia existentes, estudamos o algoritmo RSA, pois ele é muito utilizado em aplicações comerciais e foi o primeiro algoritmo assimétrico mundialmente adotado como padrão (BARBOSA *et al*, 2003).

São diversas aplicações, como por exemplo, no software *Skype* e o único algoritmo assimétrico que pode ser utilizado no Brasil, por entidades que emitem certificados digitais, as Autoridades Certificadoras (AC), estas são auditadas pela AC- Raiz. Em 28 de junho de 2001, pela medida provisória 2200/01, o Governo Federal instituiu a Infra-Estrutura de Chaves Públicas Brasileira (ICP-Brasil), com o dever de fornecer condições adequadas para conferir validade jurídica aos documentos eletrônicos.

O RSA foi desenvolvido no *Massachussets Institute of Technology* (MIT) em 1978 por Rom Rivest, Adi Shamir e Leonard Adleman, e batizado com as iniciais de seus nomes. É matematicamente baseado na Teoria dos Números, principalmente nas áreas de Aritmética Modular (do relógio) e Primalidade. Os resultados aqui utilizados e as demonstrações matemáticas dos mesmos estão disponíveis em Santos (2000).

A ideia do algoritmo RSA concentra-se no fato de que, embora seja fácil encontrar dois números primos de grandes dimensões (mais do que 100 dígitos), o tempo estimado para fatorar números, por exemplo, de 308 dígitos, com os algoritmos clássicos é de aproximadamente 100 mil anos (OLIVEIRA *et al*, 2003). De fato, ele mostra-se computacionalmente inquebrável com números de tais dimensões, e a sua força é geralmente

quantificada com o número de *bits* utilizados para descrever tais números. Para um número de 100 dígitos são necessários cerca de 350 *bits*, e as implementações atuais superam os 512 e mesmo os 1024 *bits*.

Os passos do processo utilizando o método RSA: o destinatário escolhe dois números primos extensos, $p$ e $q$. Definimos $n = p.q$, chamado de módulo, e outro inteiro positivo $e$ que deve ser relativamente primo com $\phi(n)$, valor da função de Euler[3], que corresponde a dizer que $e$ é inversível módulo $\phi(n)$. Em geral, o número $e$ é escolhido como um número primo. O par de números $(n,e)$ forma a chave pública, que deve ser enviada ao (s) remetente (s) para que ele (s) possa (m) cifrar as mensagens.

O remetente, utilizando a chave pública recebida do destinatário $(e,n)$, obtém a mensagem cifrada $c$ calculando o resto da divisão entre $m^e$ e $n$, que deve ser enviada para o destinatário. O destinatário decifra a mensagem $c$ e encontra $m$ calculando o resto da divisão entre $c^d$ e $n$. Por não ser objetivo deste trabalho não iremos demonstrar essa propriedade, mas a mesma está disponível em Silva (2006, p. 21). Em seguida converte a mensagem $m$ para letras através do código **ASCII**, recuperando assim a mensagem original.

Para criptografar uma mensagem, o remetente converte cada letra dela em um número inteiro positivo. Essa conversão é chamada de pré-codificação, para que o computador possa entender. A escolha desses números pode variar, mas geralmente utiliza-se o código **ASCII**, o mesmo utilizado pelos computadores para representar os caracteres. Depois esses números são agrupados, formando um número único que chamaremos de $m$. Se $m$ for maior que $n$, então ele deve ser dividido em blocos onde cada um seja menor que $n$.

Devemos ter o cuidado de verificar se $m<n$. Agora se calcula um inteiro positivo $d$ tal que o resto da divisão entre $e.d$ e $\phi(n)$ seja igual a 1. O par de números $n$ e $d$ forma a chave privada, que o destinatário deverá manter em segredo.

Para um melhor esclarecimento sobre os passos do algoritmo RSA, segue abaixo um exemplo utilizando valores pequenos para $p$, $q$, $e$ e $d$.

1. Tomando os primos $p = 11$ e $q = 13$, temos $n = 11.13 = 143$. Seja $e = 13$ que é primo com $\phi(n) = (11-1).(13-1) = 120$. Assim a chave pública é o par 13 e 143 que deve ser enviado para o remetente. Para a chave privada temos $d = 37$, pois $37.13 = 481 \equiv 1 \pmod{120}$. Portanto a chave privada é o par (37,143), devendo a mesma ser mantida em segredo;

[3] Lê-se fí de $n$, $\phi(n) = \#\{x \in \{1,n-1\}: \text{mdc}(x,n)=1\}$, isto é, o número de inteiros de 1 até $n$-1 que são primos relativos com $n$. Quando $n$ só possui dois fatores primos diferentes, $p$ e $q$, tem-se $\phi(n) = (p-1).(q-1)$.

2. O remetente deseja enviar a mensagem ‘RSA’. Para isso ele converte as letras R, S e A para seus respectivos códigos **ASCII**: 82, 83 e 65. Agrupando-os temos a mensagem pré-codificada $m$ = 828365. Como $m$ é maior que $n$, vamos separá-la em 3 blocos: 82, 83 e 65.
3. Para criptografar a mensagem o remetente utiliza a chave pública, $e$ = 13 e $n$ = 143, e para cada bloco calcula o resto da divisão entre $m^e$ e $n$, após as operações matemáticas, obtendo a mensagem cifrada $c$ = 41865 que será enviada ao destinatário.
4. Para o destinatário decifrar a mensagem $c$ é feito novamente à separação em blocos menores que $n$ e utilizando a sua chave privada $d$ =37 e $n$ = 143, calcula o resto da divisão entre $c^d$ e $n$, recuperando a mensagem $m$ = 828365.
5. Convertendo os valores numéricos, veja tabela 3, para letras temos a mensagem original ‘RSA’.

Tabela 3: Cifragem e decifragem da mensagem ‘RSA’

| **Cifragem** | | | | | **Decifragem** | | | |
|---|---|---|---|---|---|---|---|---|
| | $m^e$ | (mod $n$) ≡ | $c$ | | $c^d$ | (mod $n$) ≡ | $m$ | |
| R | $82^{13}$ | (mod 143) ≡ | 4 | **Transmissão → C=41865** | $4^{37}$ | (mod 143) ≡ | 82 | R |
| S | $83^{13}$ | (mod 143) ≡ | 18 | | $18^{37}$ | (mod 143) ≡ | 83 | S |
| *A* | $65^{13}$ | (mod 143) ≡ | 65 | | $65^{37}$ | (mod 143) ≡ | 65 | A |

Fonte: Dados da pesquisa, 2007.

Os cálculos acima podem parecer complicados, mas o RSA200 com módulo de 200 dígitos (663 *bits*), onde

n= 7997833911221327870829467638722601621070446786955428537560009929326128400107609345671052955360856061822351910951365788637105954482006576775098580557613579098734950144178863178946295187237869221823983. (RSA Laboratories, 2007)

fatorado em 9 de maio de 2005 por Bahr, Boehm, Franke e Kleinjung.

## 2.3 SEGURANÇA E DESEMPENHO DO RSA

A segurança desse sistema se baseia na dificuldade de fatorar um número inteiro muito grande, pois muitas operações são envolvidas no processo. De acordo com Silva (2006), vários métodos foram estudados por Gauss, Euler e Fermat com intuito de fatorar números inteiros, mas ainda não existe um método eficiente quando os números são enormes. Para se ter uma idéia, o método General Number Field Sieve (GNFS), atualmente o algoritmo mais eficiente, fatorou em 1994 um número de 129 dígitos utilizando uma rede de 1600 computadores em menos de 8 meses (BARBOSA *et al*, 2003).

A tentativa de recuperar a chave privada é chamada 'de ataque'. Os ataques comuns ao RSA são: a Força Bruta, Ataques Matemáticos e Ataques Temporais (BARBOSA *et al*, 2003).

Uma estratégia usada para quebrar a cifragem de uma mensagem, o ataque de força bruta consiste em tentar todas as combinações de chaves possíveis até conseguir decifrar a mensagem. Quando a chave é muito grande é necessário muito poder de processamento para utilizar todas as combinações em um tempo mais curto. Por isso o ataque de força bruta se torna muito difícil. Quanto ao ataque matemático, podem ocorrer as seguintes possibilidades:

1. Fatorar $n$ em fatores primos para encontrar p e q e assim calcular $\phi(n) = (p\text{-}1)\,(q\text{-}1)$ e em seguida $d$;
2. Encontrar $\phi(n)$ testando a primalidade de 1 até $n$ para calcular d;
3. Calcular $m$ diretamente fazendo $m = \sqrt[e]{c} \pmod{n}$.

Entretanto, esses métodos são inviáveis na prática, pois exigem muito poder computacional devido ao enorme tamanho de $n$. Portanto, para se ter um RSA seguro é preciso escolher os números primos $p$ e $q$ bem grandes para dificultar a fatoração de $n$ e, também, escolher tamanhos maiores para as chaves privada e pública. Na perspectiva de Barbosa *et al* (2003), o tamanho das chaves deve ir aumentando à medida que os algoritmos de fatoração e os computadores ficam mais rápidos. Corroborando com este pensamento Fuzitaki (2004) pondera que aumentará exponencialmente, também, o número de cálculos para cifrar e decifrar as mensagens.

## 3 METODOLOGIA

Para realizarmos a análise entre segurança e velocidade, vamos implementar o algoritmo de criptografia RSA, utilizando a linguagem de programação C, o auxílio da

biblioteca *GNU Multiple Precision Arithmetic Library* (*GMP*) para a manipulação de números inteiros extensos, distribuída gratuitamente sob a licença de software livre *GNU Lesser General Public License*, versão 4.4.2 de 30 de agosto de 2007.

A implementação do algoritmo RSA foi executada em um computador com as seguintes características: Processador AMD Athlon(tm) XP 2000+ com clock de 1,66GHz e 608MB de memória RAM, sistema operacional Windows XP Professional. Desativamos todos os processos desnecessários do Windows para termos menos interferências nas medidas dos tempos e utilizamos o compilador MingW.

Atualmente, para Barbosa *et al* (2003), a maioria dos módulos utilizados na criptografia RSA, para um mínimo de segurança, possui tamanho a partir de 1024 *bits* (308 dígitos), por isso, além deste, vamos utilizar nos testes valores como 2048 *bits* (616 dígitos) e 4096 *bits* (1233 dígitos).

O algoritmo implementado realizou os seguintes procedimentos: Para cada tamanho do módulo citado acima, foi gerado aleatoriamente 20 (vinte) chaves de tamanho aproximado ao módulo utilizado e, para cada chave, ciframos e decifamos 50 (cinquenta) vezes uma mensagem de 100 (cem) caracteres aleatórios[4]. No total tivemos 1000 (mil) cifragens e 1000 (mil) decifragens, em que o tempo de cada processo foi gravado em um arquivo para posteriormente ser importado para *Excel* e os dados serem analisados, interpretados. Após calculamos a média aritmética dos mesmos em cada módulo.

Posteriormente efetuamos um segundo teste, seguindo os mesmos procedimentos citados acima, entretanto utilizando mensagens de tamanho próximo ao limite de cada módulo testado para estimarmos a velocidade média da cifragem e decifragem em cada módulo. Dessa forma, utilizamos uma mensagem de 102 caracteres para o módulo de 1024 *bits*, 205 caracteres para o módulo de 2048 *bits* e 410 caracteres para o módulo de 4096 *bits*.

Utilizamos na nossa implementação o Algoritmo de Euclides para calcular o Máximo Divisor Comum (MDC) de dois números inteiros, a versão estendida do Algoritmo Estendido de Euclides[5], para gerar a chave privada e a Potência Modular para cifrar e decifrar. Consta no anexo 1 um exemplo real de chave pública e no anexo 2 os códigos das principais funções em C da implementação do RSA.

**Algoritmo:** Extensão do Algoritmo de Euclides

[4] O número de iterações aqui utilizado tem como base a implementação feita por Paixão (2003).
[5] O Algoritmo Estendido de Euclides é uma das formas de se encontrar o máximo divisor comum (MDC) de dois números inteiros como uma combinação linear inteira destes dois inteiros, sendo utilizado para calcular o inverso modular.

ENTRADA**:** inteiros não negativos *a* e *b.*

SAÍDA**:** inteiros *x, y, d,*onde d = MDC (*a,b*) = *ax+by*.

Se $b = 0$ então Retorne ($a$,1,0).
$x_2 \leftarrow 1$; $x_1 \leftarrow 0$; $y_2 \leftarrow 0$; $y_1 \leftarrow 1$;
enquanto $b > 0$ faça
$q \leftarrow a \; div \; b$; $r \leftarrow a - qb$;
$x \leftarrow x_2 - qx_1$; $y \leftarrow y_2 - qy_1$;
$a \leftarrow b$; $b \leftarrow r$; $x_2 \leftarrow x_1$; $x_1 \leftarrow x$;
$y_2 \leftarrow y_1$; $y_1 \leftarrow y$;
d ← a; x ← $x_2$; y ← $y_2$; .
Retorne (*d,x,y)*

Não foram computados os tempos de inicialização de variáveis, geração das chaves ou outros processos que não sejam de cifragem e decifragem, sendo assim computado somente o tempo da etapa de exponenciação modular. Não entraremos em detalhes sobre os códigos e a programação propriamente dita da linguagem C e das funções da biblioteca GMP, pois não é nosso objetivo, para um maior aprofundamento veja Schildt (1997) e GNU MP (2007).

## 4 ANÁLISE DOS DADOS

Os resultados dos tempos médios em milisegundos das cifragem e decifragem para os módulos de 1024, 2048 e 4096 *bits*, utilizando uma mensagem de 100 caracteres estão apresentados na Tabela 4:

Tabela 4: Tempos de cifragem e decifragem de uma mensagem de 100 caracteres.

| Módulos | Cifragem | Decifragem |
|---|---|---|
| 1024 | 11,9 | 11,9 |
| 2048 | 73,8 | 74,0 |
| 4096 | 466,6 | 466,1 |

Fonte: Elaborado pelos autores, 2007.

Nas Figuras 1 e 2 abaixo apresentamos os gráficos dos valores da Tabela 4 para a cifragem e decifragem respectivamente. Podemos perceber nos gráficos a evolução exponencial que ocorre com o tempo dos processos à medida que aumenta o tamanho da chave, tal como afirma Fuzitaki (2004).

Figura 1: Tempos de cifragem em milisegundos de uma mensagem de 100 caracteres para os módulos de 1024, 2048 e 4096 *bits*.


Fonte**:** Dados da pesquisa, 2007.

Figura 2: Tempos de decifragem em milisegundos de uma mensagem de 100 caracteres para os módulos de 1024, 2048 e 4096 *bits*.


Fonte: Elaborado pelos autores, 2007.

Na Tabela 5 apresentamos os resultados dos tempos médios em milisegundos das cifragens e decifragens utilizando, para cada módulo, uma mensagem de tamanho próximo ao limite do mesmo.

Tabela 5: Tempos de cifragem e decifragem utilizando mensagens de 102, 205 e 410 caracteres.

| Módulos (bits) | Mensagem (carac.) | Cifragem (ms) | Decifragem (ms) |
|---|---|---|---|
| 1024 | 102 | 11,9 | 12,1 |
| 2048 | 205 | 74,4 | 73,7 |
| 4096 | 410 | 466,3 | 465,5 |

Fonte: Dados da pesquisa, 2007.

Podemos observar na Tabela 5 que os valores pouco diferem da Tabela 4 mesmo para mensagens de tamanhos maiores. Notamos também que utilizando o módulo de 2048 *bits* podemos cifrar, em um único bloco, o dobro do tamanho suportado pelo módulo de 1024 *bits*, porém aquele não apresenta melhor desempenho que este, pois podemos cifrar 612 caracteres (6 vezes mais) em 6 blocos de 102 caracteres em menos tempo, utilizando o módulo de 1024 *bits*. O mesmo raciocínio pode ser feito com os demais módulos, demonstrando que podemos criptografar mensagens maiores e mais rápidas com módulos pequenos, abrindo mão de uma segurança maior.

Considerando o aumento linear do tempo de ciframento e deciframento em relação ao aumento do tamanho da mensagem, apresentamos na Tabela 6 a velocidade média estimada de cada módulo, em caracteres por segundo. E para uma melhor visualização desses valores, temos nas Figuras 3 e 4, os gráficos que exibem respectivamente a velocidade de cifragem e decifragem.

Tabela 6: Velocidade média estimada das cifragens e decifragens.

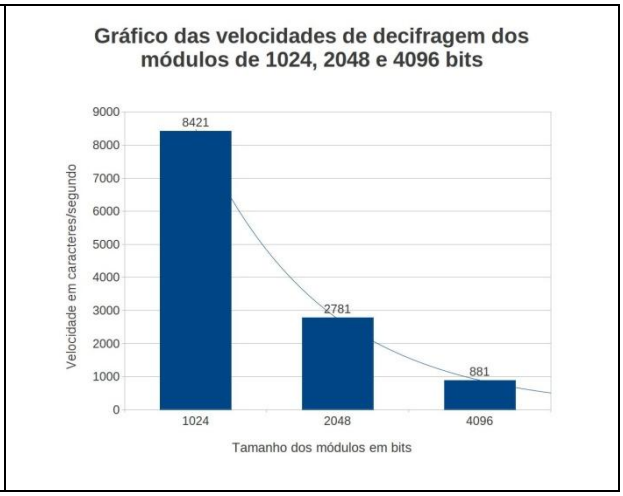
| Módulos (bits) | Cifragem (carac/seg) | Decifragem (carac/seg) |
|---|---|---|
| 1024 | 8608 | 8421 |
| 2048 | 2757 | 2781 |
| 4096 | 879 | 881 |

Fonte: Dados da pesquisa, 2007.

Figura 3: Velocidade média estimada das cifragens em caracteres por segundo para os módulos de 1024, 2048 e 4096 *bits*.


Fonte: Dados da pesquisa, 2007.

Figura 4: Velocidade média estimada das decifragens em caracteres por segundo para os módulos de 1024, 2048 e 4096 *bits*.




Fonte: Dados da pesquisa, 2007.

Observamos então que a velocidade das criptografias e descriptografias diminuem também exponencialmente conforme aumenta o tamanho do módulo, o que vai ao encontro do entendimento de FUZITAKI que explica: "[...] aumentar o valor de *n* também aumentará exponencialmente a quantidade de cálculo necessária para encriptar e decriptar" (FUZITAKI, 2004, p. 53).

## 5 CONSIDERAÇÕES FINAIS

Conforme os testes realizados, percebemos que a busca pela segurança através do aumento do tamanho das chaves provoca um aumento exponencial no tempo das cifragens e decifragens, o que pode trazer resultados negativos. Em um servidor de Internet, por exemplo, onde circula um volume enorme de informações, exigirão alto poder de processamento para utilizar a criptografia RSA de forma rápida.

Dessa forma, a escolha do tamanho da chave deverá levar em conta o grau de importância e o tamanho da informação que se queira proteger. Se a informação for de alta relevância, como um arquivo que contém as senhas bancárias dos correntistas, precisa-se de uma criptografia mais forte devido à importância das informações. Mas se for de baixa relevância, como arquivo pessoal, pode-se melhor criptografá-la com chaves menores, por que será mais rápido (o que será mais rápido). Embora o módulo de 1024 *bits* seja menos seguro que o de 4096 *bits* é necessário ainda um grande esforço computacional para fatorá-lo.

Por isso, de acordo com Benits Júnior (2003), atualmente as cifragens e decifragens são realizadas geralmente pelos algoritmos simétricos, que são mais rápidos, e o transporte das chaves é feito através do sistema assimétrico.

Contudo, o que destruiria o RSA seria a criação de um algoritmo de fatoração eficiente ou outro método de encontrar *d* sem utilizar a fatoração de *n*. Neste caso, para Barbosa *et al* (2003), a saída seria utilizar criptografia baseada em curvas elípticas, pois utilizam grupos e polinômios mais complexos. Nesta direção, Silva (2006) afirma que existe uma pesquisa sobre a construção de um computador quântico, capaz de fatorar números e encontrar primos com mais velocidade, o que daria início a Criptografia Quântica.

## RSA ENCRYPTION ALGORITHM:
## analysis between safety and speed

## ABSTRACT[6]

This work aims to discuss the existent relationship among the search for the safety of the data and the coding/ decoding speed of the RSA encryption algorithm (The name RSA came from the last name of the first writers Ron Rivest, Adi Shamir and Leonard Adleman), which uses a pair of integer numbers as "key". Considering the size of the key as requirement to safety, owing the computational difficulty for factoring extensive integer numbers, we simulated these processes, with the algorithm implemented in the C programming language, using random keys of 1024, 2048 and 4096 *bits*. This way, we observed the time of processing according the keys size, confronting safety and performance.

**Keywords:** Encryption. RSA algorithm. Security. Performance.

## REFERÊNCIAS

BARBOSA, Luis Alberto de Moraes *et al*. **RSA:** Criptografia Assimétrica e Assinatura Digital. 2003. 50 p. (Especialização em Redes de Computadores) - Universidade Estadual de Campinas, Campinas, 2003.

BENITS JÚNIOR, Waldyr Dias. **Sistemas criptográficos baseados em identidades pessoais.** 2003. 213 p. Dissertação (Mestrado em Ciência da Computação) - Universidade de São Paulo, São Paulo, 2003.

[6] Revisão realizada por Kênya Karoline Ribeiro Sodré (CRLE – Revista **Eventos Pedagógicos**).

BRAGA, Pedro Henrique da Costa. **Técnicas de Engenharia Social**. Universidade Federal do Rio de Janeiro. GRIS, 2011.

COUTINHO, Severino Collier. **Números Primos e Criptografia RSA**. IMPA, 2000.

FARIA, Fabiano Otávio de. **Estudo da técnica de Criptografia Algoritmo posicional - Alpos na segurança de dados de um banco de dados**. 2006. 56 p. Trabalho de conclusão de curso (Bacharelado em Sistemas de Informação) - Faculdades Santo Agostinho, Montes Claros, 2006.

FUZITAKI, Gerson Yoshio. **Avaliação de Desempenho de Algoritmos de Criptografia**. 2004. 85 p. Trabalho de conclusão de curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas, Universidade Estadual de Londrina, Londrina, 2004.

GNU MP. **The GNU Multiple Precision Arithmetic Library**. 2007.

OLIVEIRA, Breno Guimarães. **Fundamentos da Criptologia Parte I – Introdução e Histórias.** Universidade Federal do Rio de Janeiro, GRIS, 2005.

OLIVEIRA, Breno Guimarães. **Fundamentos da Criptologia Parte II – Criptografia Simétrica.** Universidade Federal do Rio de Janeiro, GRIS, 2005.

OLIVEIRA, Ivan S. *et al*. Computação Quântica: Manipulando a informação oculta do mundo quântico. **Ciência Hoje**, v. 28, p. 74, 2003.

PAIXÃO, Cesar Alison Monteiro. **Implementação e análise comparativa de variações do criptossistema RSA.** 2003. 173 p. Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2003.

SANTOS, José Plínio de Oliveira. **Introdução à teoria dos números.** 2. ed. Rio de Janeiro: IMPA, 2000. (Coleção Matemática Universitária).

SCHILDT, Herbert. **C, completo e total.** Tradução e revisão: Roberto Carlos Mayer. 3. ed. São Paulo: Pearson Makson Books, 1997.

RSA Laboratories, **RSA-200 is factored!**.
Disponível em: <http://www.rsa.com/rsalabs/node.asp?id=2879>. Acessado em: 10 mar. 2007.

SILVA, Elen Viviani Pereira da. **Introdução à Criptografia RSA.** 2006. 32 p. - Faculdade de Engenharia de Ilha Solteira, Universidade Estadual Paulista 'Julio de Mesquita Filho', Ilha Solteira, 2006.

## ANEXO 1

Exemplo de uma chave pública comercial utilizada pelo GnuPG apresentada como um arquivo de texto comum, mas escrito em Hexadecimal.

-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v1.0.6 (GNU/Linux),
Comment: For info see http://www.gnupg.org

mQGiBD6cuEoRBACQ6QKhCjN2qKHmOeeOdW9wOnnnd9V0eLREreSmMsRD6kSdXnr
GLQGrXMCBjelwOKB/vh2mcKn646PmSaq4sC+bLSOMAhME/IaDyDWZNWXo37yYlhD
uVD5wZGZNGMwov/bPDvjNjJTSXlo4/glbUNyVU2n9kiFYoDoSGjg1jQcYLwCgyeOJtHF
SoelA62s+YmypI3KAzkcD/1S7Fim4p5X6HA/mUkFtuExDggba+OKmsmYNfPZ/Q0sPAK8
5Or0zKAbGXAKUqezCuKCZ6FaesbIU5hTcQb2zKmzg1hzmtsWCTrxuj2X8c//yRjDzFjT8
KKCYKZWBGQTapGhvlHHq8ZsLlYZRzYyYtZ13a809FWk8G0Aq5/FAbe3KA/4nK+XA
ZUwXqXzu+xKINMu98zi9QWnVfBA5G132uCyrwOTBG1BV3803QFxGXdBrEnGlz/RU9
xjkVnEBWyqriO+lGDXuEZ6pWyx70UJ1S2qPDYKxoLTB68ZWAfKUxYP8JYBC06hk6Zt
q7FjkQGHxKMYQ9HCC3l9octyNsux+b/5Q+rQhU2lsdmlvIFJoYXR0byA8cmhhdHRvQHJ
pc2V1cC5uZXQ+iFcEExECABcFAj6cuEoFCwcKAwQDFQMCAxYCAQIXgAAKCRDA5
viFRw5zwno5AKCGTBDCw9dalpr+SG7MQx00u1iehwCeKuGC1FljdhHcu2CQX/JZoWLB
/Qi5AQ0EPpy4TRAEAKmsLjxI2k/ITXu2kZJ7+SQPftb9yRs8FxOsnmytkUwO8HaryPfuoM
U51xG8XYL4blGL6u2J67KJOO4R3buwUDmH16RC+nmMNlnaa0zlozsYIuB+r3s18hNLA
ss1LX0P0Ob6Ownar5VM8yNgVhEZkwBs6VhVlfinYThWmpaXXLU3AAMFA/498Mfl1rs
4z6vzkmIGu3Mqy+2CXSA/oCp9zPfFLJNM+WUGhpbxkbbsNEzHdTFWPcoHi23k01KjT5
CAiyiP30o6g8OV+3/WRYqeR4UNT6e87JDZo8kzjnTigI7XoAkqTJhL8pzhzvjboGZAaN1L
DPgO2H//iRaBUjjrGaOTl9x2c4hGBBgRAgAGBQI+nLhNAAoJEMDm+IVHDnPCsrYAoL
9sLObVCteWmkAPFL3b5e/pUFfAAKCzRRY3tPu6sHczFzOcw3SzeDN5x5kBogQ+nLhKE
QQAkOkCoQozdqih5jnnjnVvcDp553fVdHi0RK3kpjLEQ+pEnV56xi0Bq1zAgY3pcDigf74d
pnCp+uOj5kmquLAvmy0jjAITBPyGg8g1mTVl6N+8mJYQ7lQ+cGRmTRjMKL/2zw74zYy
U0l5aOP4JW1DclVNp/ZIhWKA6Eho4NY0HGC8AoMnjibRxUqHpQOtrPmJsqSNygM5HA
/9UuxYpuKeV+hwP5lJBbbhMQ4IG2vjiprJmDXz2f0NLDwCvOTq9MygGxlwClKnswrigme
hWnrGyFOYU3EG9syps4NYc5rbFgk68bo9l/HP/8kYw8xY0/CigmCmVgRkE2qRob5Rx6v
GbC5WGUc2MmLWdd2vNPRVpPBtAKufxQG3tygP+JyvlwGVMF6l87vsSiDTLvfM4vUF
p1XwQORtd9rgsq8DkwRtQVd/NN0BcRl3QaxJxpc/0VPcY5FZxAVsqq4jvpRg17hGeqVsse
9FCdUtqjw2CsaC0wevGVgHylMWD/CWAQtOoZOmbauxY5EBh8SjGEPRwgt5faHLcjbLs
fm/+UPq0IVNpbHZpbyBSaGF0dG8gPHJoYXR0b0ByaXNldXAubmV0PohXBBMRAgAX
BQI+nLhKBQsHCgMEAxUDAgMWAgECF4AACgkQwOb4hUcOc8J6OQCghkwQwsPXp
aa/khuzEMdNLtYnocAnirhgtRZY3YR3LtgkF/yWaFiwf0IuQENBD6cuE0QBACprC48SNp
PyE17tpGSe/kkD37W/ckbPBcTrJ5srZFMDvB2q8j37qDFOdcRvF2C+G5Ri+rtieuyiTjuEd27
sFA5h9ekQvp5jDZZ2mtM5aM7GCLgfq97NfITSwLLNS19D9Dm+jsJ2q+VTPMjYFYRGZ
MAbOlYVZX4p2E4VpqWl1y1NwADBQP+PfDH5da7OM+r85JiBrtzKsvtgl0gP6Aqfcz3xSy

TTPllBoaW8ZG27DRMx3UxVj3KB4tt5NNSo0+QgIsoj99KOoPDlft/1kWKnkeFDU+nvOy
Q2aPJM4504oCO16AJKkyYS/Kc4c7426BmQGjdSwz4Dth//4kWgVI46xmjk5fcdnOIRgQY
EQIABgUCPpy4TQAKCRDA5viFRw5zwrK2AKC/bCzm1QrXlppADxS92+Xv6VBXwACs
0UWN7T7urB3MxcznMN0s3gz
ecc=
=1J0v
-----END PGP PUBLIC KEY BLOCK-----

**ANEXO 2**

**Código das principais funções em C da implementação do RSA**

```
// Gera números Aleatórios "n" de "b" bits
// para gerar os primos e a chave privada
gmp_randstate_t state;
gmp_randinit_default (state);
gmp_randseed_ui(state, rand());
mpz_urandomb (n, state, b);
gmp_randclear(state);
// Calcula n = p*q
mpz_mul(n,p,q);
// Calcula a fun¸c~ao de Euler fi(n)=(p-1)(q-1)
mpz_sub_ui(s1,p,1);
mpz_sub_ui(s2,q,1);
mpz_mul(fn,s1,s2);
// Algoritmo de Euclides Estendido para gerar a chave privada "d"
// "inv" <- a' tal que A*a' = 1(mod b)
while(mpz_cmp_ui(B,0)>0)
{
mpz_fdiv_qr(q, r,A,B);
mpz_mul(p, q,x1);
mpz_sub(x, x2, p);
mpz_set(A, B);
mpz_set(B, r);
mpz_set(x2,x1);
```

```
mpz_set(x1,x);
}
if(mpz_cmp_ui(x2,0)<0) mpz_add(x2, x2,b);
mpz_set(inv,x2); // "inv" recebe "x2
// Calcula a Potencia modular para cifrar e decifrar
// result <- aê mod m
mpz_fdiv_qr(exp, r, exp, dois);
if(mpz_cmp_ui(r, 1)==0)
mpz_set(b, a);
while(mpz_cmp_ui(exp,0)!=0) // Enquanto "m" for diferente de 0
{
mpz_fdiv_qr(exp, r, exp,dois); // Converte gradativamente "m"
mpz_mul(A,A,A);
// para binário
mpz_fdiv_r(A,A,m);
if(mpz_cmp_ui(r,1)==0) {
mpz_mul(b,A,b);
mpz_fdiv_r(b,b,m);
}}
mpz_set(result,b); // "result" recebe "b"
// Transforma os caracteres da mensagem para o código ASCII
// Forma um número "m" contendo os caracteres em ASCII
for(i=strlen(s)-1; i>=0; i--)
{
mpz_mul_ui(m,m,1000);
j=s[i];
if(j<0) j=j+256;
mpz_add_ui(m,m,j);
}
// Transforma o código ASCII em caracteres
while(mpz_cmp_ui(M,1000)>0)
{
MensDec[k]=mpz_fdiv_q_ui(M, M, 1000);
k++;
```

```
}
//
Algoritmo de Euclides - verifica se dois números s~ ao primos entre si
//
Se MDC(a,b) = 1 retorna 1, Se MDC(a,b) != 1 retorna 0
do {
mpz_fdiv_r(r, A,B);
mpz_set(A, B);
mpz_set(B, r);
} while(mpz_cmp_ui(B,0)!=0);
if(mpz_cmp_ui(A,1)==0) {
// Se A=1 retorna 1
return 1;
}
else {
// Se A!=1 retorna 0
return 0;
}
```