REVISTA DO CCEI
ISSN Impresso: 1415-2061
ISSN Online: 2356-6635
Centro de Ciências
Economia e Informática
Revista Técnico-Científica
URCAMP
EDIURCAMP

# UM ESTUDO COMPARATIVO DE ALGORITMOS DE CRIPTOGRAFIA DE CHAVE PÚBLICA: RSA VERSUS CURVAS ELÍPTICAS

**Luis Eduardo Ferreira[1], Vinicius Ribeiro[2], Sidnei Silveira,[3] Jorge Zabadal[4]**

**Resumo**: Este trabalho apresenta um estudo comparativo entre as técnicas de criptografia RSA e Curvas Elípticas. Um protótipo de software foi desenvolvido para executar e comparar ambas as técnicas a fim de medir aspectos de desempenho computacional no que diz respeito ao emprego de memória, CPU e outras variáveis de interesse, entre as duas implementações. Estes parâmetros foram coletados e organizados a fim de elaborar a análise destes algoritmos. Os resultados apontam melhores condições para emprego do RSA em algumas situações – como no emprego de pequenas quantidades de dados –, ao passo que em outras – como a preocupação com o desempenho –, ECC apresenta melhor desempenho.

**Palavras-chave**: Criptografia de Chave Pública, Criptografia de Curvas Elípticas, RSA, Segurança Computacional

## 1. INTRODUÇÃO

Dados de conteúdo sigilosos constantemente precisam ser compartilhados entre um ou vários destinatários. O problema deste compartilhamento é a facilidade com que estes podem ser interceptados durante a transmissão e usados para fins diferentes dos quais foram originalmente concebidos. Nestes casos, existe a necessidade de se criptografar os dados a fim de garantir a segurança e privacidade destes dados, até que estes sejam recebidos por seu destinatário.

A criptografia de dados surgiu da necessidade de se assegurar não apenas a privacidade dos dados, mas também a veracidade de seu destinatário. Este trabalho aborda a implementação do tipo de criptografia que utiliza chaves pública e privada para encriptar e decriptar os dados, garantindo que apenas os destinatários pertinentes dos dados consigam acessá-los.

Para fim de estudo acadêmico, foi realizado em um primeiro momento um planejamento de como se utilizar de um protótipo de sistema para a execução e estudo de resultados das técnicas de criptografia RSA e Curvas Elípticas. Os objetivos principais deste trabalho são a descrição de como foi concebido este protótipo e a análise dos resultados obtidos.

Ambos os algoritmos escolhidos são implementações de criptografia usando chaves públicas. O grande diferencial está na forma, tamanho e processamento necessário para a geração destas. Geralmente, estes processos são baseados em problemas matemáticos conhecidos: enquanto RSA trabalha com o problema de Fatoração de Inteiros, a implementação de criptografia com Curvas Elípticas, escolhida para este trabalho, se baseia no problema do logaritmo discreto em curvas elípticas (ECDLP - *Elliptic Curve Discrete Logarithm lProblem*).

Os desafios e motivações deste trabalho se encontram em como apresentar os dados de maneira pertinente afim de obter um melhor embasamento na escolha do algoritmo a ser utilizado em soluções de criptografia e na complexidade de implementar um protótipo que faça uso de ambas as técnicas. Os resultados obtidos levaram em consideração parâmetros de consumo de memória, tamanho do arquivo cifrado, tempo de encriptação e decriptação das mensagens.

A estrutura do presente trabalho está disposta da seguinte forma: na seção 2, é apresentado o referencial teórico. Na seção 3, o detalhamento do protótipo é comentado. Resultados e análise e são apresentados na seção 4, e as considerações finais são colocadas na seção 5.

## 2. REFERENCIAL TEÓRICO

Esta seção apresenta o detalhamento das técnicas de criptografia para desenvolvimento do trabalho, como Criptografia de Chave Pública e Privada, RSA e Curvas Elípticas.

### 2.1 Criptografia de Chave Pública e Privada

Em 1976, Whitfield Diffie e Martin Hellman mudaram o paradigma da criptografia com o sistema de criptografia de chave pública (SCHNEIER, 1996; STALLINGS, 2003). Anteriormente, a criptografia era feita com o sistema de chave única, utilizando algoritmos simétricos. Alguém que possuísse a chave de encriptação poderia criptografar e decriptografar uma mensagem sem muito esforço. Apenas conhecendo esta chave ou utilizando-se de força bruta poder-se-ia obter a mensagem cifrada.

Algoritmos que usam sistemas de Chave Pública necessitam de um par de chaves, pública e privada, para troca de mensagens. A chave pública serve para criptografar a mensagem e qualquer um que possua esta chave pode criar uma mensagem criptografada. Para decriptografar mensagem é necessário o uso da chave privada, que é computacionalmente difícil de ser deduzida possuindo-se apenas a chave pública. (SCHNEIER, 1996)

Assim sendo, a cifragem de mensagens utilizando chaves pública e privada é apresentada abaixo:

| |
|---|
| 1. Alice e Bob concordam em um sistema de criptografia por chave pública; |
| 2. Bob envia sua chave pública para Alice; |
| 3. Alice cifra a mensagem utilizando a chave pública de Bob e envia para ele mensagem encriptada; |
| 4. Bob decriptografa a mensagem de Alice utilizando a sua chave privada. |

Quadro 1. Troca de mensagens utilizando Chave Pública (SCHNEIER, 1996)

Para um retorno de mensagens bastaria ser executado o mesmo processo, utilizando agora a chave pública de Alice para criptografar a mensagem e a chave privada de Alice para descriptografá-la.

## 2.2 RSA

O algoritmo RSA foi desenvolvido por Ron Rivest, Adi Shamir e Leonard Adleman. Entre suas maiores qualidades estão o fato deste ser facilmente compreendido e de sua eficácia como sistema criptográfico. (SCHNEIER, 1996)

A segurança do RSA está na dificuldade de se fatorar grandes números. As chaves pública e privada são geradas a partir de um par de grandes números primos, chamados de $p$ e $q$. Estes, de preferência, devem possuir uma grande quantidade de dígitos (100 ou mais) para assegurar a geração de chaves eficazes. Aconselha-se também que $p$ e $q$ tenham o mesmo tamanho. Com posse destes dois números, calcula-se o valor da variável $n$, utilizada como parte das chaves pública e privada. (SCHNEIER, 1996)

$$n = pq$$

Em seguida, escolhe-se aleatoriamente um valor para a variável $e$, tal que:

1. $e$ seja primo;
2. $1 < e < ((p\text{-}1)(q\text{ -}1))$;
3. $e$ seja diferente de $p$ e $q$.

Finalmente, com posse de todas as variáveis necessárias, calcula-se o valor da variável $d$ através do Algoritmo Estendido de Euclides, conforme fórmula abaixo:

$$d = e^{-1} \bmod ((p\text{-}1)(q\text{-}1))$$

O par de números ($n$, $e$) constituem a chave pública do algoritmo de RSA e o par ($n$, $d$) a chave privada. De posse dessas chaves, podem-se aplicar as duas fórmulas abaixo para encriptar ou decriptar uma mensagem $m$ (sendo $m$ um bloco numérico menor que $n$):

1. Encriptação: $c = m^e \bmod n$
2. Decriptação: $m = c^d \bmod n$

A seguir, são apresentados exemplos de parâmetros de RSA utilizados no protótipo desenvolvido (valores em hexadecimal):

1. n:
```
8e108f396ddcc6750607d02629291827a358acd61fa3b6be67bb5b12a95c231bc17a71209bb2c
47c6ba79a1a8952c20f09411a1d22f2caf6ec686128781d2132f53f34b8a04b5fff03d0cf8d21
9ba2058310295475dcc7ed311e4d0e983961fd3757446332d29629fb1f425ca683d234cfd67fe
695494f7c68131865f62c06d3
```

2. e: 10001

3. d:
```
70cea56c2cd8ca9dd214122cbb326e83cb2976d4f07ac7799a55239445f76bc976710bb3a0eee
b2701352d21353a09ee76a2c361af86abb437350b315f6f821405d8f7e65fd090c0645546b2a2
ad06abfb96151f49770a0fdb733cb7caa7d5ec070a33bea4443272306854e45f806b14b6488cf
0487ea0b6dca3626fa04d01c1
```

4. P:
```
9797b735ea095e1888879e0514e4bfc87337b54b30b71b8ec9a49f4cedce9f5679335ce52f033
d3da0952d167f2f93fa63e9ef9be71d25fb1f9b236d7ae46f15
```

5. Q:
efe8e884ae5a535bede5b0d7da7ffa5183da1a482a50d3e2cdf054cd00b91299b1297a98398dd
4264eff04a4b354089a340deb96f632419017e4ef03689a5847

Com base nestes valores, é possível fazer o uso das formulas de encriptação e decriptação acima mencionadas.

## 2.3 Criptografia utilizando Curvas Elípticas

Com o uso crescente de sistemas criptográficos e da capacidade de processamento dos computadores atuais, as chaves de criptografia baseadas em RSA têm crescido consideravelmente em tamanho. Este crescimento tende a aumentar o custo de processamento e tamanho dos dados criptografados. Sistemas que utilizem este tipo de segurança acabam por ter um aumento muito grande em tráfego de dados, podendo causar congestionamentos de rede e lentidão nos serviços. (STALLINGS, 2003)

Ainda segundo Stallings(2003), o principal atrativo para o uso de curvas elípticas para criptografia, em comparação ao RSA, é que este algoritmo aparenta proporcionar um mesmo nível de segurança utilizando chaves menores, o que solucionaria o problema do aumento de processamento e tráfego de dados e possibilitaria maiores níveis de segurança com chaves menores. Um sistema de criptografia utilizando ECC (do inglês, *Elliptical Curve Cryptography*) precisa da definição de qual curva elíptica será utilizada, do seu ponto gerador G e de um número primo $n$ relativamente grande (na ordem de $2^{180}$). O ponto G é definido como $n\mathrm{G} = 0$. A curva selecionada e o ponto gerador são valores públicos para os utilizadores do sistema. Uma abordagem simples para criptografar e decriptar as mensagens desejadas é a geração de uma tabela contendo os pontos existentes na curva selecionada, onde cada ponto corresponderia a um caractere (ou valor) da mensagem. Para a geração de chaves, deve-se escolher um número inteiro $n_A < n$. Este será a chave privada. A chave pública é gerada a partir deste número, utilizando a regra $P_A = n_A G$. Como G é o ponto gerador da curva, a chave pública também será um ponto. Considerando dois indivíduos de posse de ambas as chaves pública e privada, o processo de troca de chaves procede da seguinte forma:

- A gera uma chave secreta $K = n_A P_B$

- B gera uma chave secreta $K = n_B P_A$

Ambos os valores de K são iguais, visto que:

$$n_A P_B = n\mathrm{A}(n_B \mathrm{G}) = n_B(n_A \mathrm{G}) = n_B P_A$$

Neste modelo, o indivíduo A envia uma mensagem para B. Para encriptar a mensagem, utiliza-se o seguinte processo:

$C_m = \{n_A G, P_m + n_A P_B\}$, onde

$C_m$ = mensagem, ou caractere, criptografada

$P_m$ = mensagem, ou caractere, original

Mesmo B não possuindo a chave privada de A, ambos possuem a mesma chave secreta. Logo, para se extrair a mensagem cifrada utiliza-se o seguinte método:

$$P_m + n_A P_B - n_B(n_A \mathrm{G}) = \mathrm{Pm} + n_A(n_B \mathrm{G}) - n_B(n_A \mathrm{G}) = \mathrm{Pm}$$

A segurança deste sistema se dá pelo fato de que é consideravelmente difícil calcular $n_A$ tendo apenas G e $n_A$G. Abaixo seguem exemplos de parâmetros de domínio de ECC utilizados no protótipo desenvolvido (valores em hexadecimal):

- Curva: $y^2 = x^3 + ax + b$
  - a = 340e7be2a280eb74e2be61bada745d97e8f7c300
  - b = 1e589a8595423412134faa2dbdec95c8d8675e58
- Ponto G:

(bed5af16ea3f6a4f62938c4631eb5af7bdbcdbc3,
1667cb477a1a8ec338f94741669c976316da6321)

- Primo *n*: `e95e4a5f737059dc60dfc7ad95b3d8139515620f`

O algoritmo RSA é tido como referencia para criação de sistemas criptográficos devido ao alto grau de segurança fornecido (RSA LABORATORIES, 2011). Este é amplamente usado em aplicações que fazem o uso de assinatura digital, identificação e RFID[1]. Sua grande aceitação também se da pelo fato deste atuar no mercado há vários anos e sua eficácia estar fortemente comprovada.

A divisão de segurança da empresa EMC2 foi fundada para criar soluções de segurança e padrões de sistemas de criptografia baseados em RSA (como o PKCS#1[2], RSA-OAEP[3] e RSA-PSS[4]). Hoje ela é tida como a principal fonte de patentes e soluções envolvendo o algoritmo RSA.

Surgindo pouco tempo depois que o RSA, sistemas criptográficos utilizando curvas elípticas demoraram a serem utilizados comercialmente. O alto custo computacional e poucos casos de sucesso conhecidos impediram a popularização inicial deste algoritmo. Adiciona-se a isso o fato de seu antecessor já estar amplamente divulgado e aceito pela comunidade de segurança.

Curvas Elípticas começaram a aumentar sua popularidade com a crescente necessidade de redução de tráfego de dados e segurança de chaves (ROSING, 1999; CERTICOM, 2011). A alta segurança do ECC com chaves de tamanho pequeno tornam este algoritmo interessante para ser aplicado em hardwares de segurança, que normalmente contam com pouco espaço de memória interna e necessitam de fortes criptografias.

## 3. DESENVOLVIMENTO DO PROTÓTIPO

Para ser realizada a coleta de dados necessária para a realização de um estudo comparativo se fez necessário o desenvolvimento de um protótipo. Este software deveria ser

[1] RFID,do inglês Radio Frequence Identification, Identificação por Radiofrequência

[2] PKCS#1, do inglês Public-Key Cryptography Standard, Padrão de Criptografia de Chave Publica

[3] RSA-OAEP, do inglês Optimal Asymmetric Encryption Padding, Complemento Otimizado para Encriptação Assimétrica

[4] RSA-PSS, do inglês Probabilistic Signature Scheme, Esquema de Assinatura Probabilística

capaz de executar ambos os algoritmos, para encriptar e decriptar mensagens de texto, e medir parâmetros relativos ao tempo de execução, memória utilizada no processo e tamanho dos arquivos criptografados que foram gerados.

## 3.1 Arquitetura

O desenvolvimento deste protótipo foi realizado em linguagem Java, utilizando a IDE Eclipse Indigo[5]. Visando um melhor desempenho e mais fácil compreensão do código optou-se por não se utilizar de uma interface gráfica, deixando a execução do software rodando no plano de fundo do computador a ser usado como teste.

O software desenvolvido foi baseado na seguinte premissa: dado um certo numero X de arquivos de entrada e uma certa quantidade Y de tamanhos de chaves pré-definidos, executar o programa Z vezes e coletar todos os dados necessários durante a execução dos passos de encriptação e decriptação. A figura 1 ilustra como as classes Java ficaram distribuídas a fim de se chegar ao resultado desejado:

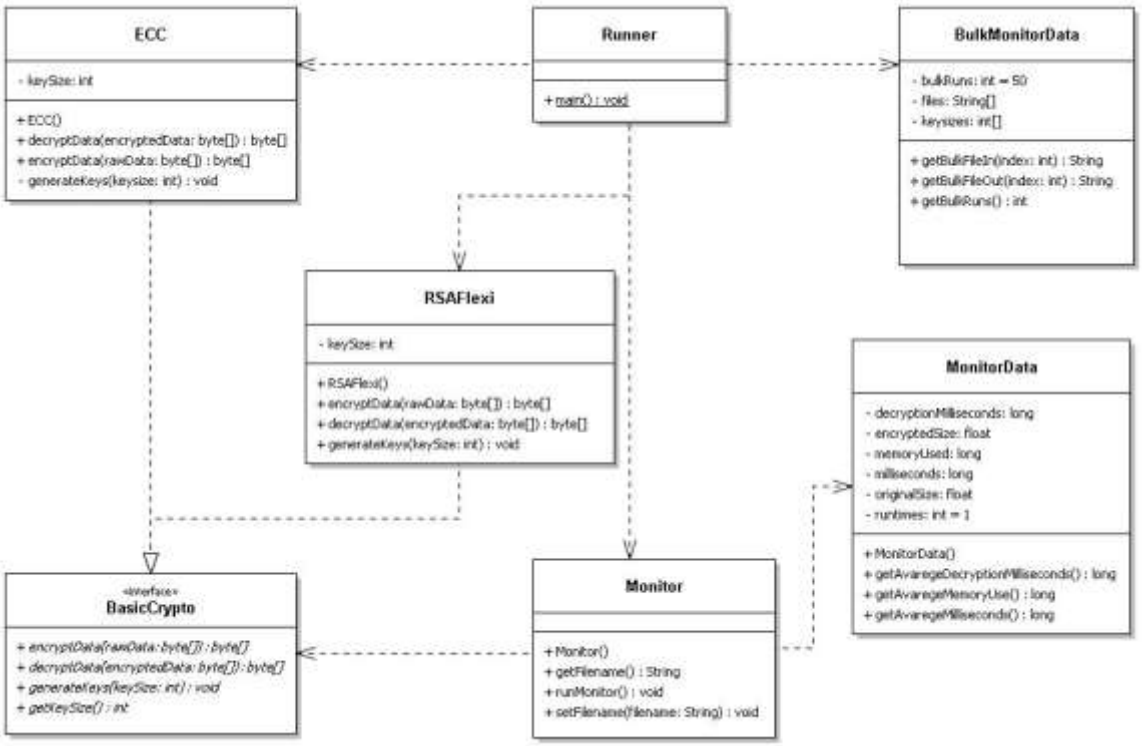


Figura 1 - Diagrama de Classes

A classe *BulkMonitorData* é responsavel por manter os dados de quantidade de execução, lista de arquivos de entrada a serem usados e a lista de chaves à serem utilizadas durante todo o

[5] IDE, do inglês, Integrated Development Environment, ambiente de desenvolvimento integrado. A versão Indigo é a mais atual, sendo otimizada para desenvolvimento Java versão 7.

experimento. Esta classe é chamada no início do processo para que estes parâmetros fossem utilizados de mesma forma para a execução do algoritmo RSA e ECC.

Para o desenvolvimento deste trabalho foram definidos os seguintes parâmetros:

- 13 arquivos de entrada serão utilizados, cada um contendo um texto gerado aleatoriamente. Estes arquivos contêm, respectivamente, os tamanhos de 10 Bytes, 1K, 5k, 10k, 20k, 50k, 100k, 300k, 500k, 1MB, 2MB, 5MB, 10MB e 30MB;
- 7 tamanhos de chaves pré-definidos, sendo estes 160bits, 192 bits, 224 bits, 256 bits, 320 bits, 384 bits e 512 bits;
- cada combinação *arquivo x chave* foi executada 50 vezes, e as médias dos resultados foram armazenadas para análise posterior.

Os arquivos de entrada foram escolhidos tendo como base tua utilização. Arquivos menores caracterizam senhas fracas e fortes e os arquivos maiores textos a serem criptografados. Os tamanhos de chave foram selecionados com base na lista de chaves disponíveis na biblioteca de classes utilizada no desenvolvimento da classe de Curvas Elípticas.

As classes *RSAFlexi* e *ECC* são responsáveis pela geração das chaves (procedimento que não é parte das medições necessárias para este trabalho), cifragem e decifragem dos arquivos. Com base nisto a classe *Runner*, responsável pela execução do protótipo, cria uma instancia da classe *Monitor* usando como entrada o algoritmo desejado, e os parâmetros definidos em *BulkMonitorData*.

Durante a execução de *Monitor*, a classe *MonitorData* coleta os dados de tempo, tamanho e memória utilizadas e no fim calcula a média de todas as 50 execuções. Estes valores são armazenados em um arquivo de *log* para posterior análise.

## 3.2 Bibliotecas Utilizadas

A modelagem inicial contava com o uso do *security provider*[6] Bouncy Castle(2011) para a codificação da classe ECC. Para RSA seriam utilizadas as ferramentas nativas da linguagem JAVA.

Para desenvolver o código de criptografia com Curvas Elípticas uma *API*[7] desta biblioteca é disponibilizada a fim de possibilitar ao desenvolvedor utilizá-la de forma correta.

[6] Security Provider, biblioteca de classes e configurações de segurança que contém especificações de segurança como chaves de criptografia válidas, números primos passiveis de serem usados para sistemas criptográficos, parâmetros de domínio para curvas elípticas, inteiros com bits suficientes para boas gerações de chaves

[7] Application Programming Interface, interface de programação de aplicação, conjunto de métodos e classes disponibilizados pela biblioteca para sua utilização.

Ocorreu que, durante o desenvolvimento do protótipo, as instruções de uso desta API não estavam disponíveis. Sem esta referência tornou-se impraticável o uso da Bouncy Castle(2011).

Como alternativa para este trabalho foi encontrada o security provider FlexiProvider (FLEXIPROVIDER, 2011) A biblioteca Flexi possui em seu conjunto de ferramentas tanto classes para desenvolvimento utilizando Curvas Elípticas quanto RSA. Este conjunto de ferramentas tornou o protótipo mais robusto e o código mais legível.

## 4. ANÁLISE DE RESULTADOS

Após a execução de todos os testes configurados no protótipo, os resultados foram alinhados em tabelas de dados a fim de se conseguir elaborar a análise entre os dois algoritmos e conseguir diferenciá-los em seus pontos fortes e fracos.

O primeiro dado a ser levado em consideração foi o resultado final da cifragem dos dados. Os resultados ilustrados na figura 2 com valores listados em kilobytes, demonstram um pequeno aumento no tamanho final do arquivo quando este é criptografado utilizando RSA. Essa diferença começa a ficar mais significativa quando utilizamos chaves com tamanho de 512bits. Percebe-se, nas figuras 2 e 3 que, quando os dados são processados utilizando ECC, podem-se notar dois comportamentos distintos. Conforme o tamanho de chave cresce, o tamanho do arquivo final também é incrementado, mesmo em pequenas proporções. Outro comportamento leva em conta o tamanho do arquivo gerado no final do processo: este diminui proporcionalmente, chegando a casos em que o tamanho do arquivo cifrado chega a ser menor que o tamanho do arquivo original. Este comportamento pode ser notado inicialmente em arquivos com 10k (tamanho original exato: 10,063K, tamanho processado exato: 10,056K), onde o resultado é menor do que os 10 K.

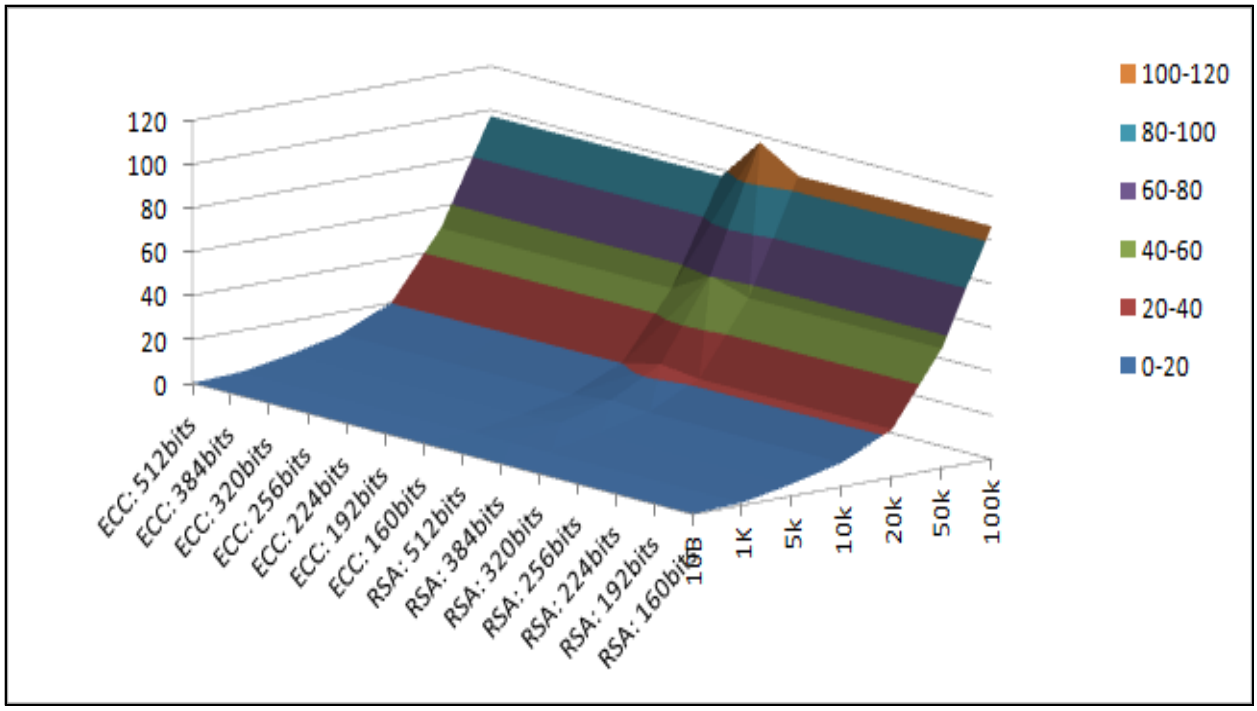


Figura 2 – Processamento de arquivos de 5b à 100k

Tamanho de Arquivo Original (K) x Tamanho de Arquivo Cifrado (K)

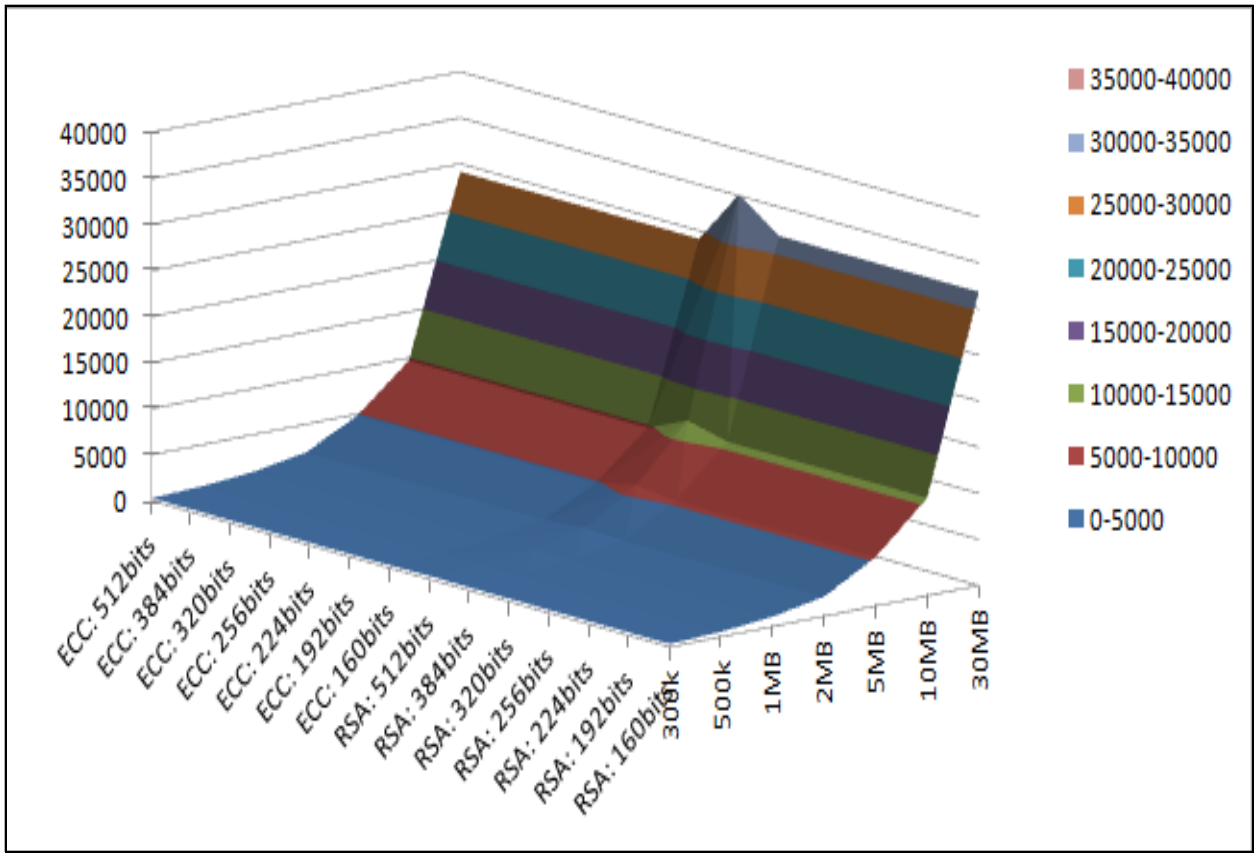


Figura 3 Processamento de arquivos de 300 à 30MB

Tamanho de Arquivo Original (K) x Tamanho de Arquivo Cifrado (K)

O segundo parâmetro a ser analisado leva em consideração os tempos de execução de cada processo de encriptação e decriptação dos dados. As figuras 4 a 7 lustram a evolução do tempo de encriptação, em milissegundos, dos algoritmos RSA e ECC.

Conforme as figuras 4 a 7 enquanto o tamanho do arquivo aumenta o tempo de encriptação do RSA cresce consideravelmente, ultrapassando em grande escala os tempos de execução da criptografia com ECC. Mesmo sendo mais veloz em tamanhos de arquivos menores, este comportamento pode ser notado a partir de arquivos com 5k.

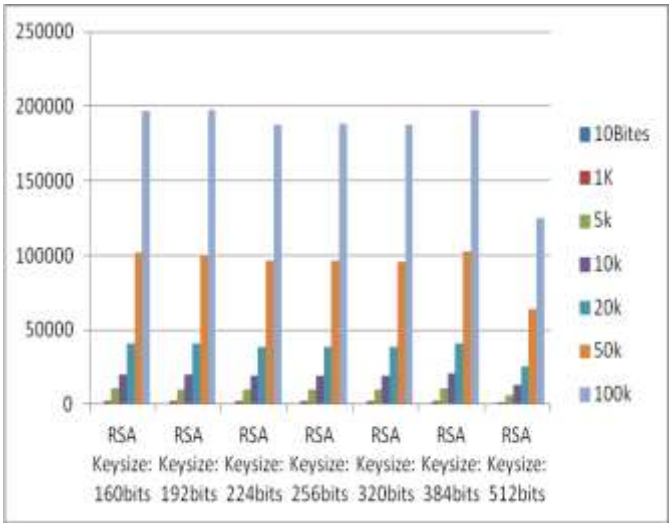


Figura 4 Tempo de Encriptação, em milissegundos (RSA 10b - 100k)

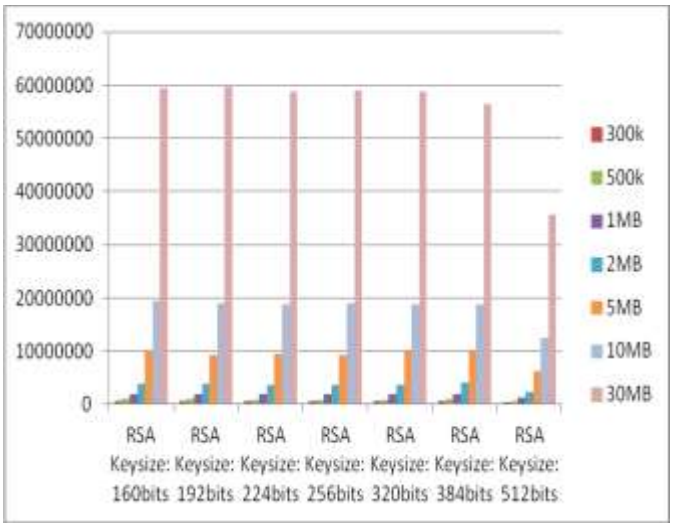


Figura 5 Tempo de Encriptação, em milissegundos (RSA 300k - 30MB)

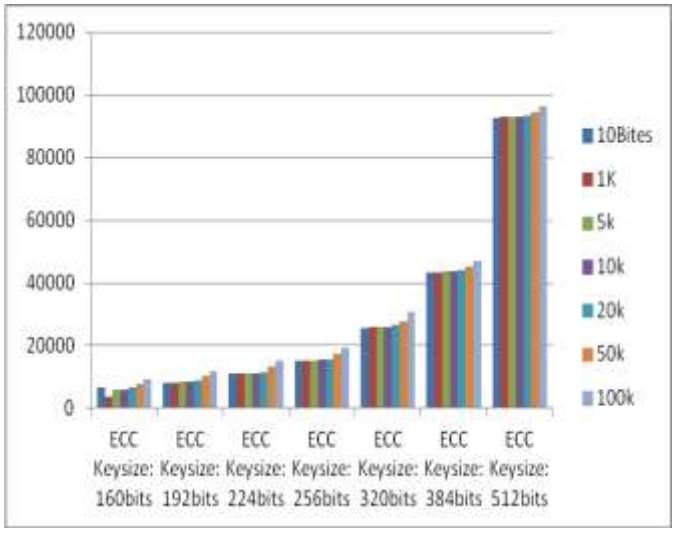


Figura 6 Tempo de Encriptação, em milissegundos (ECC 10b - 100k)

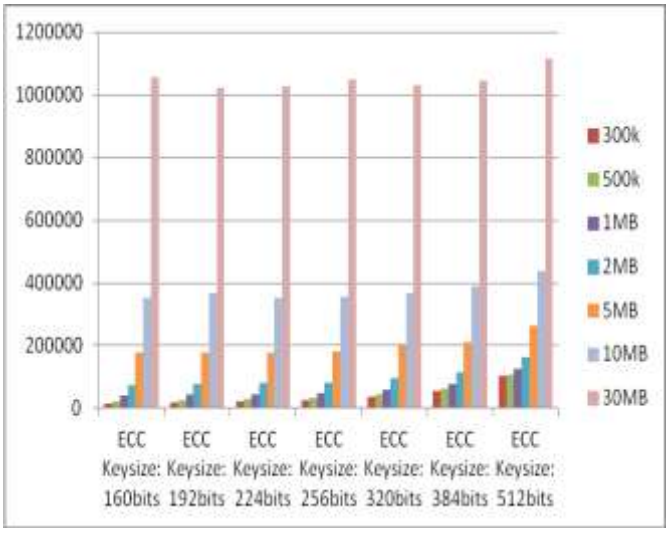


Figura 7 Tempo de Encriptação, em milissegundos (ECC 300k - 30MB)

Os resultados de tempo de execução de decriptação – figuras 8 a 11 – seguem a mesma tendência apresentada na encriptação, mas com valores maiores. Isto sugere que o processo de decriptação é oneroso para ambos os algoritmos.

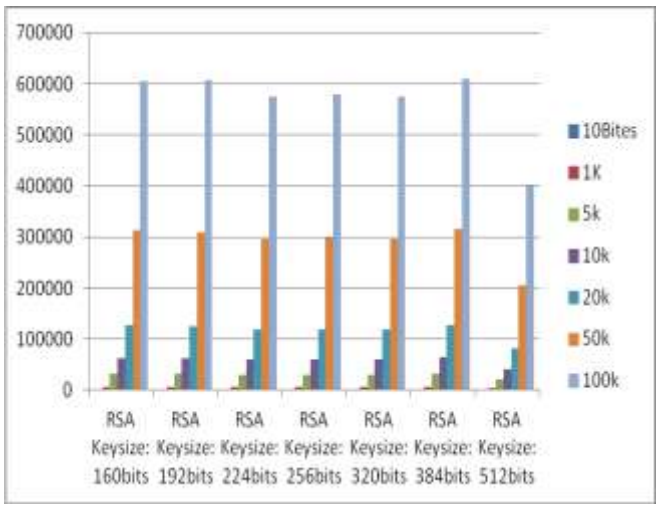


Figura 8 Tempo de Decriptação, em milissegundos (RSA 10b - 100k)

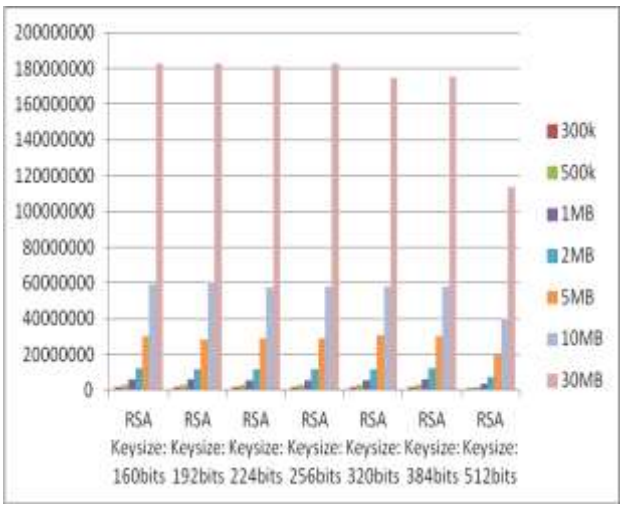


Figura 9 Tempo de Decriptação, em milissegundos (RSA 300k - 30MB)

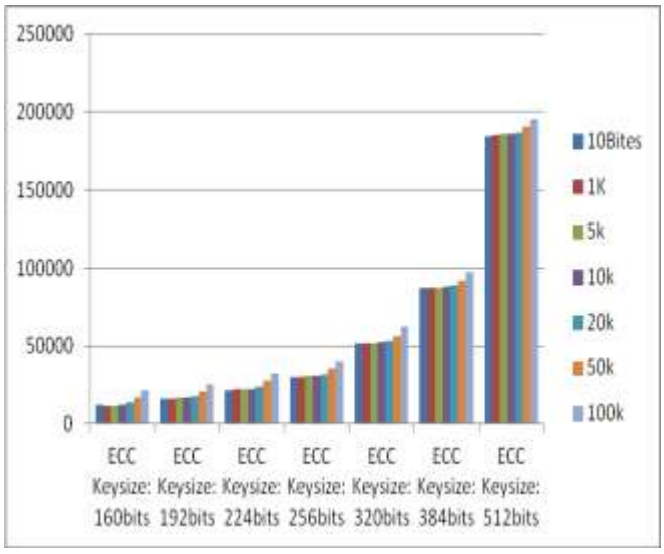


Figura 10 Tempo de Decriptação, em milissegundos (ECC 10b - 100k)

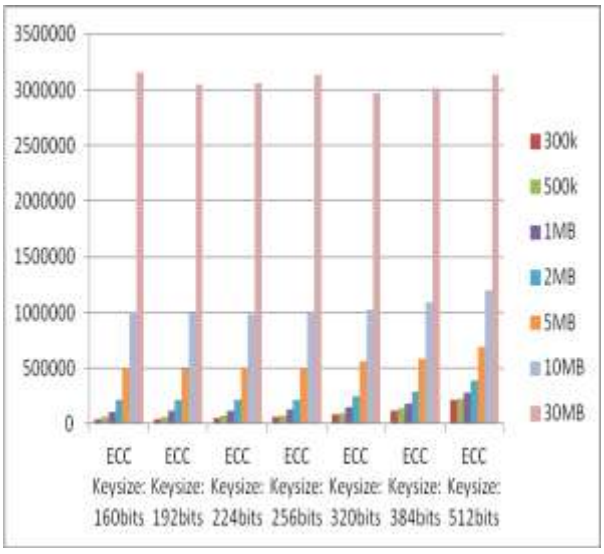


Figura 11 Tempo de Decriptação, em milissegundos (ECC 300k - 30MB)

O uso de memória, terceiro parâmetro utilizado para comparação neste trabalho, apresentou resultados mais complicados de serem analisados. As Figuras 12 e 13 ilustram os comportamentos verificados durante a execução de ambas as técnicas.

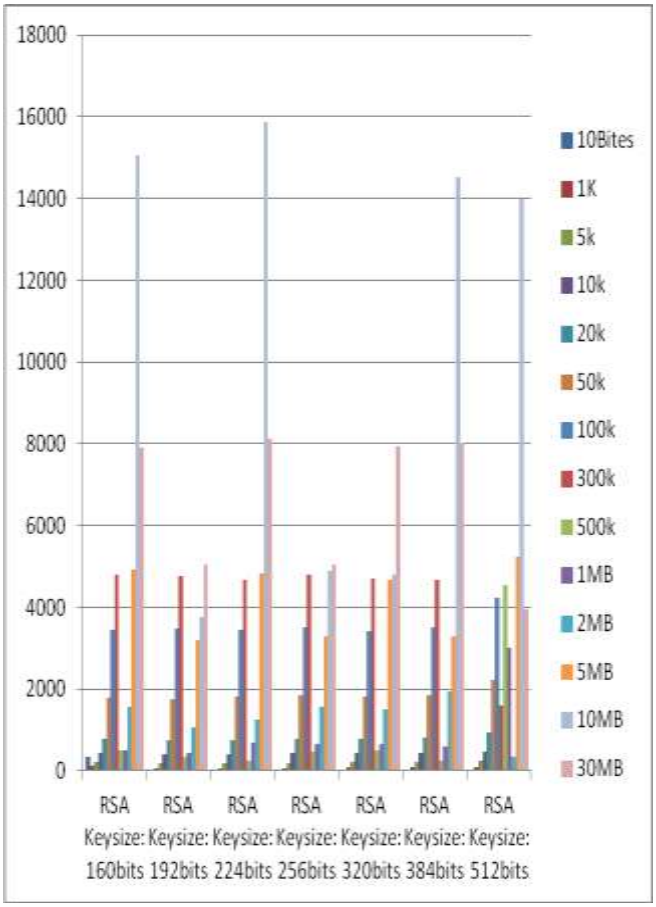


Figura 12 RSA - Consumo de Memória, em KB

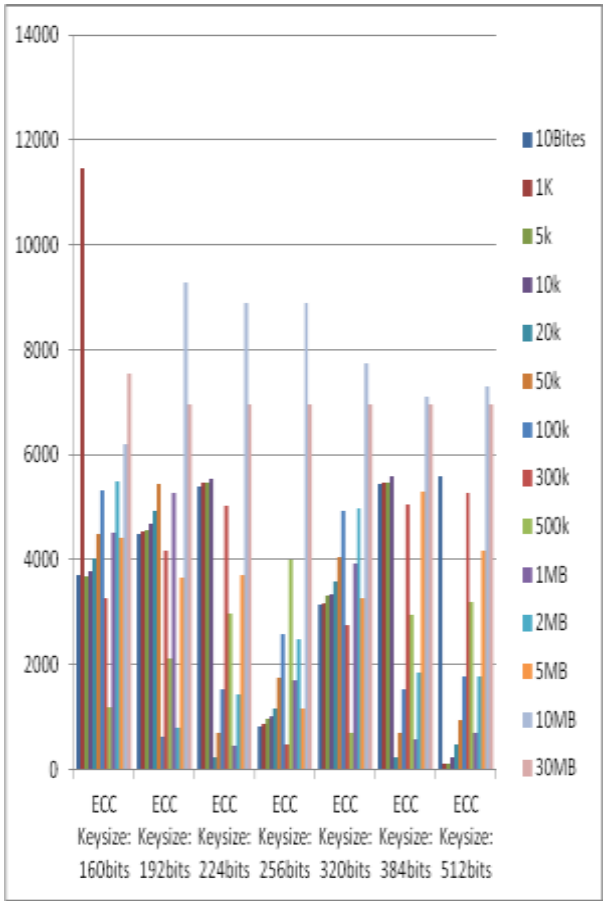


Figura 13 ECC - Consumo de Memória, em KB

Embora ECC tenha apresentado resultados mais interessantes de desempenho quanto a tempo e tamanho de arquivo gerado, a memória consumida para a execução destes processos apresentou em um geral um consumo maior de memória do processo Java.

A grande dificuldade na medição do consumo de memória se dá pelo fato de que a linguagem Java não lida diretamente com a memória do computador. Os processos Java são executados sobre JVM[8]s. Esta JVM aloca um espaço na memória principal do computador, chamada de Heap, e pode, conforme a necessidade do processo, ser aumentada e diminuída. No ponto de vista de processos rodando no computador isto é uma grande vantagem já que os processos ocupam apenas o necessário para serem executados. Sob um ponto de vista de monitoramento de processos é um problema, pois existe uma grande dificuldade de se medir efetivamente quanto cada processo Java está consumindo dentro desta Heap. Para efetuar este

[8] JVM, Java Virtual Machine. Java é uma linguagem interpretada, então seu código objeto é compilado para uma JVM única e cada fabricante desenvolve uma JVM para seu sistema. Existem JVMs para Intel, Mac, Unix,Symbian e outros sistemas operacionais.

tipo de monitoramento seriam necessárias estruturas mais robustas, como um servidor de aplicação dedicado para este objetivo.

## 5. CONCLUSÕES

Ao decidir por fazer uso de um sistema criptográfico em um sistema de computação, vários fatores devem ser levados em conta. Questões como quantidade de requisições a serem feitas ao sistema, quantidade de dados a serem protegidos e nível de segurança devem ser considerados nesta escolha. Conforme análise dos resultados obtidos com o protótipo desenvolvido, ambas as técnicas de criptografia possuem seus pontos fortes e fracos nos quesitos mencionados.

O algoritmo RSA demonstrou excelente desempenho quando utilizado em pequenas quantidades de dados. Porém, conforme os dados foram aumentando de tamanho, tanto o tempo de execução aumentou quanto o tamanho do arquivo gerado cresceu. Este tipo de comportamento pode não ser agradável quando se pensa em tráfego de dados na rede ou via internet.

A criptografia utilizando ECC desempenhou melhor resultado quando posta a condições mais severas. Em casos de médios e grandes arquivos a serem criptografados este algoritmo retornou os melhores resultados, tanto na questão de tamanho quanto no tempo de execução do processo.

Pode-se concluir que a criptografia utilizando Curvas Elípticas é recomendada quando se possui a preocupação da quantidade de dados a serem transitados ou ainda na necessidade de maior agilidade no acesso a estes dados. O uso ideal deste algoritmo se dá quando chaves de menor tamanho (mesmo estas sendo de igual ou maior segurança que chaves maiores de RSA) podem ser utilizadas e se tem em foco o alto desempenho do aplicativo gerador da criptografia.

A técnica criada por Rivest, Shamir e Adleman ainda pode ser considerada uma referência no campo da criptografia nos dias de hoje, mas conta com a característica de necessitar de tamanhos de chave cada vez maiores, a fim de prover a segurança necessária. Dependendo do tipo de sistema, este aumento de dados e de processamento pode se tornar um fator crítico a ponto de uma migração de sistema de criptografia ser necessário.

## REFERÊNCIAS

BOUNCY CASTLE. Bouncy Castle Crypto APIs. Disponível em: <http://bouncycastle.org>, acesso em Junho de 2011.

CERTICOM. Elliptic Curve Cryptography (ECC). Disponível em: <http://www.certicom.com/index.php/ecc>. Acesso em Novembro de 2011.

FLEXIPROVIDER. FlexiProvider for Java Cryptography Architecture (JCA/JCE). Disponível em: <http://www.flexiprovider.de/>. Acesso em Outubro de 2011.

ROSING, Michael. Implementing Elliptic Curve Cryptography. Greenwich: Manning Públications Co., 1999.

RSA LABORATORIES. RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1. Disponível em: <http://www.rsa.com/rsalabs>. Acesso em Maio de 2011.

SCHNEIER, Bruce. Applied Cryptography: protocols, algorithms, and source code in C. 2. ed. Nova Iorque: J. Wiley, 1996.

STALLINGS, William. Cryptography and network security: principles and practice ed. New Jersey: Prentice Hall, 2003.