UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DAINARA APARECIDA VOITECHEN

ANÁLISE E COMPARAÇÃO DE ALGORITMOS PARA CRIPTOGRAFIA DE IMAGENS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2015

DAINARA APARECIDA VOITECHEN

ANÁLISE E COMPARAÇÃO DE ALGORITMOS PARA CRIPTOGRAFIA DE IMAGENS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnóloga em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. MSc. Rogerio Ranthum.

PONTA GROSSA

2015

TERMO DE APROVAÇÃO

ANÁLISE E COMPARAÇÃO DE ALGORITMOS PARA CRIPTOGRAFIA DE IMAGENS

por

DAINARA APARECIDA VOITECHEN

Este Trabalho de Conclusão de Curso foi apresentado em 29 de outubro de 2015 como requisito parcial para a obtenção do título de Tecnóloga em Análise e Desenvolvimento de Sistemas. A candidata foi arguida pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Rogerio Ranthum
Prof.(a) Orientador(a)

_____
Geraldo Ranthum
Membro titular

_____
Gleifer Vaz Alves
Membro titular

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

Dedico este trabalho à minha mãe/madrinha
Nossa Senhora da Conceição Aparecida,
que foi o meu refúgio quando eu não
acreditava que iria conseguir.

AGRADECIMENTOS

Primeiramente à Deus por me proporcionar a chance de estar nesta universidade e concluir este curso. Também por me conceder saúde e força, e ser o refúgio nos momentos de grande dificuldade.

A minha mãe/madrinha Nossa Senhora da Conceiçao Aparecida, que de uma forma surpreendente me mostrou que o último caminho a seguir seria o da desistência.

A aquele que me incentivou a realizar o trabalho mesmo sabendo que o tempo era curto e que as pesquisas eram muitas. Aquele que acreditou e me fez acreditar, meu orientador Prof. MSc. Rogerio Ranthum. O apoio, a disponibilidade e a paciência em acompanhar o desenvolvimento foram essenciais para concluí-lo.

Ao Prof. PhD. Richard Duarte Ribeiro pela disponibilidade e grande ajuda, na correção da língua inglesa utilizada neste trabalho.

Aos meus pais Roseli e Dieroni, que com seu grande e infinito amor não mediram esforços para que eu pudesse concluir o curso de nível superior. O apoio e os conselhos foram essenciais para que eu chegasse até aqui.

A Universidade Tecnológica Federal do Paraná e seu corpo docente lotado no campus Ponta Grossa, que durante os anos de convivência, proporcionaram experiências que elevaram meu crescimento técnico, profissional e pessoal.

A minha família e meus amigos, por entenderem que os momentos de ausência eram necessários.

Enfim, a todos que contribuíram direta ou indiretamente para a realização deste trabalho e para minha formação, os quais foram muitos e não haveria espaço para aqui serem nominados.

"*Seja você quem for, seja qual for a posição social que você tenha na vida, a mais alta ou a mais baixa, tenha sempre como meta muita força, muita determinação e sempre faça tudo com muito amor e com muita fé em Deus, que um dia você chega lá. De alguma maneira você chega lá.*"

Ayrton Senna

RESUMO

VOITECHEN, A. Dainara. **Análise e comparação de algoritmos para criptografia de imagens**. 2015. 159. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2015.

Desde os primórdios da troca de informações e da corrida ao poder, fez-se necessário o envio de mensagens entre pessoas, as quais são ligadas por um propósito, sem que as demais pessoas não autorizadas fiquem sabendo qual o conteúdo dessa informação. Com isso, criou-se uma forma de proteger os dados contra acessos indevidos, chamada de criptografia (em sua tradução: ***kripto*** – "oculto, secreto, obscuro, ininteligível" e ***grapho*** – "escrita"). Esta nada mais é do que uma forma de esconder os dados a fim de que, somente as pessoas autorizadas possam ter acesso. Nos sistemas computacionais a troca de informações ocorre em uma proporção muito grande, são diversas mensagens enviadas por segundo pelo meio de comunicação (redes corporativas e/ou ***internet***). No processo de criptografia a informação é embaralhada com uma chave correspondente, esta pode ser privada ou pública. Para embaralhar a informação em sistemas computacionais, deve-se utilizar algum dos algoritmos disponíveis, o qual pode ser simétrico ou assimétrico. Os simétricos criptografam e decriptografam apenas com a chave privada, já os assimétricos utilizam a chave pública para encriptar os dados e a chave privada para decriptografar. Numa comunicação simétrica é criada uma chave para cada envio de mensagem, sendo que esta chave deve ser compartilhada pelo canal inseguro a quem necessita desta informação. Já na comunicação assimétrica, cada ponto da comunicação possui um par de chaves e apenas a chave pública de cada um é compartilhada. Os algoritmos simétricos se caracterizam pela rapidez no processo e os assimétricos pela lentidão. O escopo deste trabalho se refere ao compartilhamento de imagens de trânsito captadas pelos radares fixos, onde a imagem transita por um canal inseguro saindo do processador instalado no local até o ponto de processamento da multa. Para que esta não seja interceptada e utilizada de má fé por pessoas não autorizadas é necessário que a mesma fique ininteligível enquanto trafega pela rede, mas, que volte a ser uma imagem clara ao fim do processo. Este trabalho demonstra os desempenhos de tempo de acordo com o tamanho da chave, na criptografia de imagens, utilizando os algoritmos simétricos AES, ***Blowfish***, RC2, RC5, RC4, DES e 3DES e o assimétrico RSA, bem como demonstra o desempenho na integridade da imagem durante o processo. Foram utilizadas chaves de tamanho 128, 192 e 256 ***bits*** para os algoritmos AES, ***Blowfish***, RC2, RC5 e RC4, já para o DES e o 3DES foram utilizados tamanhos de chave fixo como descreve a especificação de cada um (64 e 192 ***bits*** respectivamente), para o algoritmo RSA foram utilizadas chaves de tamanho 2048, 4096 e 16384 ***bits***. Com isso, este trabalho conclui que o melhor método para criptografar grandes quantidades de ***bits*** (imagens) é a utilização em conjunto dos algoritmos RSA e RC5.

Palavras-chave: Criptografia. Imagens. Desempenho. Algoritmos.

ABSTRACT

VOITECHEN, A. Dainara. **Analysis and comparison algorithms for encrypting images**. 2015. 159. Term Paper (Technology Analysis and Systems Development) – Federal Technology University – Parana. Ponta Grossa, 2015.

Since the beginning of the exchange of information and the race to power, it was necessary to send messages between people, who are connected by a purpose, without unauthorized people knowing the content of that information. Thus, it was created a way to protect data from unauthorized access, called encryption (in his translation: Kripto - "hidden, secret, obscure, unintelligible" and grapho - "writing"). This is nothing more than a way to hide the data so that only authorized persons can gain access to it. In computer systems the information exchange occurs on a very large proportion and are several messages sent per second by means of communication (corporate network and/or Internet). In the encryption process the information is shuffled with a matching key, which can be private or public. To scramble the information in computer systems, one should use some of the available algorithms, which can be symmetrical or asymmetrical. In symmetric ones the encryption and decryption only happen with the private key, while the asymmetric use the public key to encrypt data and the private key to decrypt it. In a symmetrical communication a key is created for each message sent, and this key must be shared with who needs this information through the unreliable channel. In the asymmetric communication, each communication point has a key pair and only the public key of each one is shared. Symmetric algorithms are characterized by fast processing and asymmetric ones for slowness. The scope of this work refers to the sharing of traffic images, captured by fixed speed cameras, in which the image is sent through an insecure channel out of the processor installed in place until the fine's processing point. To avoid that the image would be intercepted and used in bad faith by unauthorized persons is necessary to make it unintelligible while travelling over the network, but returning again to a clear picture at the end of the process. This work demonstrates the performance time according to the key size in cryptography images using the symmetric algorithms AES, Blowfish, RC2, RC5, RC4, DES and 3DES and asymmetric RSA, and also demonstrates the performance in the image integrity during the process. It were used size keys of 128, 192 and 256 bits for the AES algorithms Blowfish, RC2, RC5 and RC4, while for DES and 3DES it were used fixed key sizes as described in the specification of each one (64 and 192 bits respectively), for the RSA algorithm were used size keys of 2048, 4096 and 16384 bits. Therefore, this paper concludes that the best method to encrypt large amounts of bits (images) is the use of RSA together and RC5 algorithms.

Keywords: Encryption. Images. Performance. Algorithms.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

3DES	Triple Data Encryption Standard
AAD	Additional Authentication Data
ADSL	Asymmetric Digital Subscriber Line
AES	Advanced Encryption Standard
API	Application Programming Interface
ARCFOUR	Alleged Rivest Ciphers Four
CBC	Cipher Block Chaining
CERT.br	Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil
CPF	Cadastro de Pessoa Física
CPU	Central Processing Unit
DES	Data Encryption Standard
DESede	Nomenclatura utilizada para Triple Data Encryption Standard
DoS	Denial of Service
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECIES	Elliptic Curve Integrated Encryption Scheme
GB	GigaByte
HD	Hard Disk
HmacMD5	Hash Media Access Control and Message Digest 5
HmacSHA1	Hash Media Access Control and Secure Hash Algorithm – 160 bits
HmacSHA224	Hash Media Access Control and Secure Hash Algorithm – 224 bits
HmacSHA256	Hash Media Access Control and Secure Hash Algorithm – 256 bits
HmacSHA384	Hash Media Access Control and Secure Hash Algorithm – 384 bits
HmacSHA512	Hash Media Access Control and Secure Hash Algorithm – 512 bits
IBM	International Business Machines
ICP-Brasil	Infraestrutura de Chaves Públicas Brasileiras
IDE	Integrated Development Environment

IDEA	International Data Encryption Algorithm
IP	Internet Protocol
IPSec	Internet Protocol Security
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JPG	Joint Photographic Group
KB	KiloByte
KSA	Key Scheduling Algorithm
LAP	Leitura Automática de Placas
MB	MegaByte
MD2	Message Digest 2
MD4	Message Digest 4
MD5	Message Digest 5
NIST	National Institute of Standards and Technology
NSA	National Security Agency
PBEWithHmac	Public Beta Envioriment com Secure Hash Algorithm – 160 bits e
SHA1AndDESede	Triple Data Encryption Standard
PBEWithMD5And	Public Beta Envioriment com Message Digest e Data Encryption
DES	Standard
PKCS	Public-Key Cryptography Standards
PGP	Pretty Good Privacy
PGPFone	Pretty Good Privacy Phone
PRGA	Pseudo-Random Generation Algorithm
RAM	Random Access Memory
RC	Rivest Ciphers
RC2	Rivest Ciphers 2
RC4	Rivest Ciphers 4
RC5	Rivest Ciphers 5
RC6	Rivest Ciphers 6
RG	Registro Geral

RGB	Red, Green, Blue
RSA	Rivest, Shamir and Adleman
s	Segundos
S/MIME	Secure Multipurpose Mail Extension
SHA-1	Secure Hash Algorithm – 160 bits
SHA-2	Secure Hash Algorithm – 256 bits
SHA-224	Secure Hash Algorithm – 224 bits
SHA-256	Secure Hash Algorithm – 256 bits
SHA-384	Secure Hash Algorithm – 384 bits
SHA-512	Secure Hash Algorithm – 512 bits
SNMP	Secure Network Management Protocol
SSL	Secure Socket Layer
SUN	Stanford University Network
TCP	Transmission Control Protocol
XOR	Ou Exclusivo
WEP	Wired Equivalent Privacy

SUMÁRIO

1 INTRODUÇÃO

A segurança da informação é um dos temas mais discutidos na atualidade. Enviar dados pelas redes de computadores (internas e externas) e/ou armazenar estes em mídias ou banco de dados e garantir que eles não serão acessados por pessoas não autorizadas é uma questão que necessita de uma atenção especial. Quanto mais se pesquisam meios de garantir a integridade da informação, mais ataques são desenvolvidos.

Um sistema de informação deve garantir a confiabilidade, a integridade, a disponibilidade e a autenticidade dos dados que trafegam por ele. Quando os dados são enviados por um canal inseguro (***Internet*** e redes corporativas) deve-se garantir que eles não serão adulterados e que serão enviados por determinado usuário (sendo que este possui a autorização para manipulação dos dados), além disto estas informações devem estar disponíveis para o conjunto de pessoas que tem o dever e/ou o direito de utilizá-los.

Os ataques realizados nos sistemas podem ser de diversos tipos, como por exemplo, por engenharia social (onde são obtidas as informações por meio da exploração da confiança de quem utiliza o sistema).

> Dentre os fatos que demonstram o aumento da importância da segurança, pode-se destacar a rápida disseminação de vírus e worms, que são cada vez mais sofisticados. Utilizando técnicas que incluem a engenharia social, canais seguros de comunicação, exploração de vulnerabilidades e arquitetura distribuída, os ataques visam a contaminação e a disseminação rápida, além do uso das vítimas como origem de novos ataques. (NAKAMURA E GEUS, 2007, p. 11).

Também podem ser realizados métodos para conseguir a informação direto no sistema (força bruta, criptoánalise, etc.). “Um ataque de força bruta, ou brute force, consiste em adivinhar, por tentativa e erro, um nome de usuário e senha e, assim, executar processos e acessar sites, computadores e serviços em nome e com os mesmos privilégios deste usuário.” (CERT.br, 2012, p. 20).

“Criptoánalise ou criptánalise. 1. Métodos de analisar mensagens cifradas com o objetivo de decifrá-las. 2. Arte ou ciência de determinar a chave ou decifrar

mensagens sem conhecer a chave. Uma tentativa de criptoánalise é chamada ataque." (MORAES, 2004, p. XXI).

Os sistemas também podem ser atacados e deixarem de estar disponíveis aos usuários (negação de serviço – isto pode gerar grandes perdas as organizações, etc.). "Negação de servicço, ou DoS (***Denial of Service***), é uma técnica pela qual um atacante utiliza um computador para tirar de operação um serviço, um computador ou uma rede conectada à ***Internet***." (CERT.br, 2012, p. 21).

Por ser uma questão complexa, muitas vezes a segurança de um sistema é implementada de forma ineficiente, gerando perdas. Nenhuma implementação é cem por cento segura, mas, algumas formas podem garantir que a informação que trafega e/ou é armazenada via sistema mantenha sua integridade e não seja utilizada de má fé por possíveis atacantes.

Informação é um conjunto de dados que define algo. Por exemplo, informações pessoais de um cliente: RG, CPF, endereço, número de telefone, filiação, etc. Em um sistema, uma informação não necessariamente denota somente de dados em texto puro, neste conjunto podem haver imagens e/ou outros arquivos que se façam necessários. Estas informações serão armazenados ou trafegados pela rede e podem ser acessados por qualquer pessoa que queira e tenha conhecimento para tal.

Uma das formas de garantir a integridade e o sigilo das informações é a criptografia. Para desenvolver esta forma de segurança, é escolhido um determinado algoritmo que demonstre bom desempenho de tempo e de integridade da informação. Para garantir a autenticidade, um dos meios é a assinatura digital dos dados.

> A criptografia é uma ciência que possui importância fundamental para a segurança, ao servir de base para diversas tecnologias e protocolos, tais como a Secure Socket Layer (SSL) e o IP Security (IPSec). Suas propriedades — sigilo, integridade, autenticação e não-repúdio — garantem o armazenamento, as comunicações e as transações seguras, essenciais no mundo atual. (NAKAMURA E GEUS, 2007, p. 16).

A implementação de um algoritmo de criptografia nada mais é do que um conjunto de funções matemáticas e lógicas que embaralham os ***bits*** dos dados. Como uma implementação equivocada de uma dessas funções pode acarretar na perca total da segurança, a linguagem de programação Java possui uma API apropriada para a criptografia, não sendo necessária a implementação completa dos algoritmos.

A segurança de todos os algoritmos se dá pela chave utilizada. Quanto maior o tamanho desta chave, maior a segurança contra ataques de força bruta. Em teoria quanto maior a chave, maior será o tempo decorrido no processo.

Um exemplo de informação em arquivo que trafega pela rede são as imagens captadas por radares de trânsito. Estas são geradas por uma câmera e processadas no meio físico instalado no local.

> As técnicas de criptografia em imagens encontram aplicação em ambientes em que imagens confidenciais, por exemplo, imagens médicas, contratos, escrituras, mapas entre outros, precisam ser armazenadas ou transmitidas através de um canal de comunicação inseguro, a ***internet***. (SILVA ET. AL, 2013, p. 02).

Elas devem ser trafegadas deste local ao ponto em que gerará a multa de trânsito, por uma rede – atualmente estas imagens trafegam pela rede para facilitar e agilizar o processo. “Para garantir que a troca de informação confidencial de imagens ocorra de forma segura pela ***Web*** a criptografia assume papel importante.” (SILVA ET. AL, 2013, p. 01).

A criptografia de imagens se difere em alguns pontos da criptografia de texto puro por causa da grande quantidade de ***bits*** a serem escondidos durante a execução do algoritmo de criptografia. Assim, faz-se necessário um estudo aprofundado do desempenho dos algoritmos simétricos e assimétricos neste caso específico.

> Uma imagem digital é composta por pequenos pontos, chamados ***pixels***. Cada ***pixel*** é composto por três bandas de cor, uma com tom de vermelho, outra verde e azul (***Red***, ***Green*** e ***Blue*** – RGB). As três cores podem apresentar tonalidades diferentes, de acordo com a sua intensidade, e combinadas podem exibir em torno de 16 milhões de cores. (SILVA ET. AL, 2013, p. 02).

De acordo com a citação anterior, é possível entender por que deve-se escolher um algoritmo para a criptografia de imagens que apresente além de um bom desempenho de tempo decorrido, um desempenho satisfatório quanto a integridade da imagem a ser criptografada.

Caso haja perdas de informações das bandas de RGB constantes na imagem, haverá distorção. No caso de imagens de radares de trânsito, isto não poderá acontecer, pois a imagem deve estar integra no fim da comunicação para que a multa possa ser gerada.

1.1 JUSTIFICATIVA

Este trabalho justifica-se pelo fato de que não foram encontrados estudos técnicos científicos que comprovem que os algoritmos simétricos disponíveis para criptografia apresentam resultados satisfatórios no momento da encriptação específica de uma imagem.

Foram encontrados apenas dois artigos técnicos – “Aplicação de técnicas de criptografia de chaves assimétricas em imagens” e “Criptografia assimétrica de imagens utilizando algoritmo RSA” – ambos apresentados por Silva et al e referentes a criptografia de chaves assimétricas, portanto, nenhum abordando a criptografia com chaves simétricas.

Estes dois artigos apresentam a comparação de desempenho entre algoritmos assimétricos, criptografando as bandas de RGB da imagem. Isto foge ao escopo deste trabalho, onde neste, é comparado se as formas para criptografia de texto puro podem ser aplicadas igualmente a grandes quantidades de ***bits*** (imagens).

Além disso, este trabalho servirá como uma referência para os técnicos responsáveis pela escolha do melhor algoritmo para criptografia de imagens, pois demonstrará com resultados claros o melhor e o pior caso de determinados algoritmos, podendo enquadrar-se os resultados à solução almejada.

Com a evolução da ***Internet*** e das novas tecnologias é necessário encontrar a melhor solução (que seja rápida e eficaz) para a segurança durante o transporte da informação, principalmente quando ela requer sigilo, autenticidade, confiabilidade e integridade.

1.2 OBJETIVOS

Este capítulo aborda os objetivos deste trabalho, detalhando a análise comparativa a ser realizada nos algoritmos propostos para criptografia de imagens de radares de trânsito.

1.2.1 Objetivo Geral

O objetivo deste trabalho é encontrar alguns algoritmos que demonstrem o melhor desempenho de tempo decorrido e integridade de imagem, os quais devem realizar a encriptação e a decriptação de uma imagem de trânsito sem a comprometer.

No momento em que a pessoa responsável por gerar a multa de trânsito receba a imagem, ela deve estar nítida e com todos os requisitos necessários.

Além disso, estes algoritmos devem apresentar um bom desempenho no tempo decorrido para o processo, bem como, devem demonstrar um resultado satisfatório quanto a velocidade x segurança no momento de criptografar uma imagem de trânsito e quanto a velocidade x qualidade no momento da decriptação.

É imprescindível ainda que os algoritmos abordados não corrompam a imagem, que a mesma seja decriptada e se apresente conforme sua original.

1.2.2 Objetivos Especificos

Serão analisados os desempenhos de alguns algoritmos simétricos e assimétricos para criptografia de imagens, em tempo decorrido para criptografia, decriptografia e a integridade do arquivo decriptografado – pois não podem haver perdas de ***pixels*** das imagens criptografadas.

Os algoritmos simétricos e assimétrico testados neste trabalho, foram escolhidos por possuírem implementação na classe ***Cipher*** (uma das classes da API de criptografia do Java) São eles: AES, Blowfish, RC2, RC5, RC4, DES, 3DES e RSA. Os testes realizados demonstrarão qual é o melhor caso e qual é o pior caso de cada um.

A classe implementa outros algoritmos assimétricos, mas, estes não permitem a criptografia de grandes quantidades de ***bits***. Então para o trabalho foi utilizado somente o RSA, no intuito de exemplificar o caso.

Serão testados em alguns algoritmos simétricos, chaves de tamanho 128 (menor), 192 (intermediária) e 256 (maior) ***bits***, pois este é o padrão utilizado pelo algoritmo AES (padrão de segurança utilizado pelo governo dos Estados Unidos).

Sendo assim, para que seja possível realizar uma comparação, serão realizados os testes com estes tamanhos de chave nos demais algoritmos que possuem tamanho de chave variável (Blowfish, RC2, RC4 e RC5) mantendo assim o padrão.

Já os algoritmos DES e 3DES possuem tamanho de chave fixo, não sendo possível manter este padrão para eles. No algoritmo assimétrico serão utilizadas chaves de tamanho 2048, 4096 e 16384 ***bits***, para que seja possível analisar o desempenho de uma chave de tamanho menor, uma de tamanho intermediário e uma de tamanho maior, seguindo o padrão dos algoritmos simétricos.

Para os testes de tempo decorrido de acordo com o tamanho da chave serão utilizadas grandes quantidades de ***bits***. Assim será possível observar que cada algoritmo possui sua particularidade, alguns tem seu melhor caso com a chave de tamanho maior, outros com a intermediária e outros com a menor.

Será possível observar que os algoritmos RC5 e RC4 são os que apresentam os melhores resultados de tempo decorrido na maioria dos testes com os três tamanhos de chave.

O RC4 é o algoritmo que tem o melhor desempenho nos testes de tamanho de imagens. Por trabalhar com fluxo ao invés de blocos ele mantém o tamanho das imagens originais, criptografadas e decriptografadas exatamente iguais.

Será possível observar também que não é vantajoso utilizar os algoritmos DES e 3DES. Além de já terem suas chaves “quebradas por força bruta”, eles não apresentam um desempenho de tempo favorável quando comparados aos demais, e mais seguros, algoritmos simétricos.

Também será demonstrado que não foi possível analisar o desempenho do algoritmo assimétrico, pois este não permite a criptografia de grandes quantidades de ***bits***, sendo viável sua utilização apenas em texto puro.

No decorrer do trabalho são comparados os resultados obtidos em cada algoritmo simétrico (com os três tamanhos de chave) na criptografia de pacotes de imagens. Para isso foram selecionados cinco pacotes, o primeiro contendo uma única imagem, o segundo contendo dez imagens, o terceiro contendo cinquenta imagens, o quarto contendo cem imagens e o quinto contendo duzentas imagens.

Será possível notar que, como no caso da variação das chaves, na comparação com a criptografia de pacotes de imagens, cada algoritmo também possui sua particularidade, em alguns casos é mais vantajoso criptografar imagem-a-imagem, em outros a vantagem está em criptografar grandes quantidades de bits (pacotes de imagens).

Será demonstrado também, que as formas encontradas para a encriptação de dados com algoritmos simétricos podem ser adotadas igualmente para a encriptação de imagens, pois não acarretará em perda de ***pixels***.

Assim, todos os algoritmos simétricos testados podem ser utilizados no caso das infrações de trânsito, pois em qualquer um deles a imagem decriptografada continua nítida e a multa poderá ser gerada.

1.3 METODOLOGIA

Neste trabalho será feita uma análise comparativa do desempenho dos algoritmos simétricos e assimétricos, implementados pela classe ***Cipher*** do Java (AES, Blowfish, RC2, RC5, RC4, DES, 3DES e RSA), no caso específico da criptografia de imagens de radares de trânsito.

Serão comparados os desempenhos de tempo decorrido de acordo com o tamanho da chave utilizada e de acordo com a quantidade de imagens a serem criptografadas. Além disso, será testado também o desempenho dos algoritmos de acordo com a integridade da imagem decriptografada e o tamanho da imagem criptografada.

Para a realização da análise comparativa, no capítulo 2 será apresentado a abordagem teórica sobre a segurança nos meios em que esta informação pode ser trafegada. Será discutido sobre redes no capítulo 2.1.1 e sobre internet no capítulo 2.1.2.

A descrição sobre o que é criptografia é demonstrada no capitulo 2.2, e a apresentação das formas de criptografia mais utilizadas: a de chaves simétrica e a de chave assimétrica é dada no capítulo 2.3. Já os algoritmos de criptografia mais

utilizados e difundidos são demonstrados no capítulo 2.4 (2.4.1 – simétricos e 2.4.2 – assimétricos).

Todo o trabalho será realizado na linguagem de programação Java, por ser uma tecnologia que apresenta alguns benefícios, como pode ser visto no capítulo 2.5. A implementação dos algoritmos de criptografia utilizados no trabalho é demonstrada no capítulo 2.6 (AES – capítulo 2.6.1; Blowfish – capítulo 2.6.2; RC2 – capítulo 2.6.3; RC5 – capítulo 2.6.4; RC4 – capítulo 2.6.5; DES e 3DES – capítulo 2.6.6; RSA – capítulo 2.6.7).

Será dada também uma breve explicação sobre a função de ***hash*** no capítulo 2.7, mas a mesma não será implementada neste trabalho (por não atender os objetivos propostos). Para encerrar o capítulo 2, a seção 2.8 apresenta a definição de criptografia de imagens.

A descrição do caso das imagens utilizadas neste trabalho (captadas por radares de trânsito) é apresentado no capítulo 3. Já os resultados obtidos nos testes de tempo decorrido de acordo com a quantidade de imagens, de acordo com o tamanho da chave, de integridade e tamanho de arquivos, em cada um dos algoritmos testados e os recursos utilizados para os testes são elencados no capítulo 4.

No capítulo 5 é apresentada a comparação do desempenho de todos os algoritmos simétricos testados neste trabalho de acordo com cada tamanho de chave utilizado e por fim, o capítulo 6 demonstra as dificuldades encontradas na execução deste, as considerações finais e o trabalho futuro proposto.

Não será implementado neste trabalho a assinatura e a certificação digital, bem como não será abordada a transferência das imagens pelo canal. O escopo é a análise do desempenho dos algoritmos na criptografia e decriptografia específica de arquivos de extensão JPG.

Nos apêndice, são demonstradas as classes em Java implementadas e utilizadas para os testes em cada um dos algoritmos.

2 REFERENCIAL TEÓRICO

Neste capítulo são abordados os vários estudos realizados sobre a segurança dos dados. São apresentadas informações sobre a segurança nos meios de transferência dos dados (redes e ***Internet***), o que é a criptografia e como ela tem sido utilizada, os dois tipos de criptografia utilizados para codificação dos dados nos meios de comunicação e alguns dos algoritmos de criptografia que são mais amplamente difundidos.

Além disso é dada uma pequena explicação sobre a função ***hash***. Também é realizada uma breve apresentação da linguagem de programação Java e a forma como os algoritmos escolhidos para os testes foram implementados. Por fim, a última seção apresenta os estudos sobre o processamento digital e a forma como foi realizada a encriptação de imagens digitais no trabalho.

2.1 SEGURANÇA

Quando ligamos a televisão ou acessamos uma página de notícias na ***Internet***, somos amedrontados pelo crescimento da criminalidade. São assaltos, homicídios, genocídios, estupros, corrupção, desvio de dinheiro, etc. Essa gama de informações é tão corriqueira quanto a população clamando por justiça e maior segurança.

Hoje em dia, uma das principais ferramentas de propagação de crimes é a ***Internet***, e é exatamente neste meio que a segurança e as legislações ainda se mostram prematuras.

A infraestrutura de rede e a informática tiveram papel importante no avanço da globalização, tanto que podemos considerar duas revoluções primordiais no crescimento mundial: A Revolução Industrial (séculos XVIII e XIX) e a Revolução Digital (de 1960 até os dias de hoje).

> Se a tecnologia da informação é hoje o que a eletricidade foi na era industrial, em nossa época, a ***internet*** poderia ser equiparada tanto a uma rede elétrica quanto ao motor elétrico, em razão de sua capacidade de distribuir a força da informação por todo o domínio da atividade humana. (SOUZA E SILVA, 2013, p. 01).

Apesar dos vários estudos e da constante preocupação, a segurança nos meios digitais ainda não consegue inibir todo o tipo de ataques. Alguns protocolos são seguidos, mas o investimento com segurança ainda é pequeno em alguns casos, tendo em vista a quantidade de formas de ataque existentes.

Segurança da informação é um problema complexo, pois quanto mais são realizados estudos para inibir as consequências, mais ataques são desenvolvidos. Quando há uma grande concentração de pessoas, a primeira coisa que os responsáveis se preocupam é com a contratação de seguranças ou o reforço da segurança já existente.

Com a ***Internet*** e as redes não deve ser diferente, ainda mais quando necessita-se enviar informações sigilosas. Os atacantes não se aproveitam somente das vulnerabilidades dos sistemas, mas também da vulnerabilidade dos usuários que os manipulam.

Conforme pode ser notado diariamente, há um grande avanço na disponibilidade de acesso à ***Internet***, e assim muitas pessoas tem a possibilidade de se comunicar e enviar dados a várias outras pessoas no mundo inteiro, ou fazer transações financeiras com as mais diversas empresas. Muitas vezes essas pessoas possuem pouco ou nenhum conhecimento sobre o quão hostil pode ser este ambiente.

Por isso, as corporações que disponibilizam serviços na ***Internet*** e/ou utilizam redes de computadores para interligar suas matrizes e filiais, devem possuir um departamento que trate somente de questões de segurança, principalmente da segurança preventiva.

> Segurança de computadores: A proteção oferecida a um sistema de informação automatizado para atingir os objetivos apropriados de preservação da integridade, disponibilidade e confidencialidade de ativos de sistemas de informação (incluindo ***hardware***, ***software***, ***firmware***, informações/dados e telecomunicações) (STALLINGS E BROWN, 2014).

Esta descrição demonstra quais são os principais objetivos a serem alcançados com a segurança da informação: integridade, disponibilidade e confidencialidade. A integridade pode ser dividida em duas: integridade dos dados (que informações só sejam alteradas de maneira especifica e autorizada) e integridade dos sistemas (o programa deve desempenhar suas funções livre de manipulação não autorizada).

A disponibilidade garante que o sistema deve estar disponível a todo momento, sem haver negação de serviços aos usuários autorizados. A confidencialidade também pode ser dividida em duas: confidencialidade dos dados (que informações confidenciais não sejam acessadas por pessoas não autorizadas) e privacidade (controle das informações – onde podem ser armazenadas e/ou compartilhadas e quem pode ter acesso a elas).

Há vários tipos de vulnerabilidade ao qual o sistema computacional está sujeito. As três situações possíveis em sistemas, descritas na citação abaixo correspondem aos conceitos descritos no parágrafo acima:

> Ele pode ser **corrompido**, de modo a operar de forma errônea ou dar respostas erradas. Por exemplo, valores de dados armazenados podem ser diferentes do que deveriam ser porque foram modificados inadequadamente; Ele pode estar **vazando**. Por exemplo, alguém que não deveria ter acesso a algumas ou a todas as informações disponíveis por meio da rede obtém tal acesso; Ele pode tornar-se **indisponível** ou muito lento. Isto é, usar o sistema ou rede torna-se impossível ou impraticável (STALLINGS E BROWN, 2014).

Para amenizar um pouco as consequências dos ataques realizados aos computadores, as corporações e até mesmo aos usuários comuns, devem ser adotadas algumas políticas de segurança. Segundo o CERT.br: “A política de segurança define os direitos e as responsabilidades de cada um em relação a segurança dos recursos computacionais que utiliza e as penalidades as quais está sujeito, caso não às cumpra”.

Existem algumas políticas especificas, tais como: política de senhas; política de ***backup***; política de privacidade; política de confidencialidade; política de uso aceitável (ou termo de uso).

Política de senha define quais são as regras na utilização dos textos para autenticação, seu tamanho mínimo e máximo, sua formatação e qual é o período para a troca deste texto.

Política de ***backup*** define quais são as formas para realização de cópia de segurança de todos os dados, sua periodicidade e as mídias que serão utilizadas para tal.

Política de privacidade define como serão tratadas (modo de armazenamento e quem poderá ter acesso) as informações pessoais de todos os envolvidos no sistema.

Política de confidencialidade define como são realizadas as trocas de informações institucionais, de cunho gerencial (entre a matriz e suas filiais) e de cunho técnico.

Política de uso define como é feita a utilização dos recursos computacionais, as responsabilidades e os direitos de quem possui acesso ao sistema.

Uma política bem definida e alguns cuidados podem amenizar as consequências dos ataques. Além disso, também existem formas de proteção dos dados, como a criptografia, a qual será discutida neste trabalho. Os capítulos abaixo abordam a segurança das informações nos principais meios de comunicação utilizados por empresa-empresa (redes de computadores) e empresa-cliente (***Internet***).

2.1.1 Redes de Computadores

Redes de computadores nada mais são do que um conjunto de computadores interligados entre si. Esta rede pode utilizar-se da ***Internet*** (rede externa) ou não (rede interna), e os dados podem trafegar por um meio físico (cabeada) ou por ondas de rádio (***wireless***).

Com o avanço da tecnologia e com a possibilidade de várias pessoas se conectarem com outras do mundo todo, houve um aumento considerável na preocupação com a segurança da transmissão de dados. Hoje, pela rede de computadores são trocadas milhões de informações sigilosas por segundo.

> Nas décadas de 70 e 80, a informática fazia parte da retaguarda dos negócios das organizações, nas quais o enfoque principal da segurança era o sigilo dos dados. Era a época dos ***mainframes***, e a proteção era voltada para os dados. Entre as décadas de 80 e 90, com o surgimento dos ambientes de rede, a integridade passou a ser de suma importância, e a proteção era feita não tendo em mente os dados, mas sim as informações. A informática fazia parte da administração e da estratégia da organização. (NAKAMURA E GEUS, 2007, p. 50).

Essas informações são um conjunto de dados de um determinado assunto, exemplo: Sistema de cadastro de clientes: informação – cadastro do cliente; dados – nome do cliente, número do CPF, número do RG, telefone, número da conta, etc. A perda ou a má utilização desses dados acarreta em um problema extremamente sério.

Antigamente quando as redes tinham conectividade com o mundo externo limitada, a preocupação com a segurança era menor. Hoje em dia com o avanço e a conexão de redes locais a redes externas cabeadas e sem cabeamento (***wireless***), e com o avanço dos meios que se conectam a essa rede (***notebooks***, ***tablets***, celulares, etc), a preocupação aumentou e a questão segurança teve (em tese) que seguir a mesma linha de avanço tecnológico.

> A proteção visa, sob esse ponto de vista, a manutenção do acesso às informações que estão sendo disponibilizadas para os usuários. Isso significa que toda informação deve chegar aos usuários de uma forma íntegra e confiável. Para que isso aconteça, todos os elementos de rede por onde a informação flui até chegar ao seu destino devem estar disponíveis, e devem também preservar a integridade das informações. O sigilo também pode ser importante e junto com a integridade e a disponibilidade formam as propriedades mais importantes para a segurança (NAKAMURA E GEUS, 2007, p. 44).

A rede pode ser interna (privada) de uma determinada organização, mas o acesso aos dados que trafegam nela (por meio de cabos e/ou ondas de rádio, no caso de redes sem fio) pode ser feito por qualquer pessoa, bem ou mal intencionada.

Em qualquer momento estas informações que estão sendo enviadas podem ser interceptadas. “As informações que trafegam pela rede estão sujeitas a serem capturadas. As senhas que trafegam pela rede estão sujeitas a serem capturadas.” (NAKAMURA E GEUS, 2007, p. 59).

> As redes ***wireless*** abriram uma brecha enorme na segurança de sistemas em rede. Isso porque os dados podem ser facilmente interceptados com algum conhecimento técnico, isso obrigou o desenvolvimento de técnicas de criptografia para tornar esse tipo de comunicação viável, não só para empresas que decidem conectar seus usuários por meio de redes sem fio, mas, também, para que os usuários domésticos possam realizar suas transações financeiras com mais segurança e privacidade. (SOUZA E SILVA, 2013, p. 02).

Muitas empresas utilizam-se da rede para manter um relacionamento ágil e eficaz com seus fornecedores, suas filiais e seus clientes, formando assim um ambiente cooperativo. E se alguma informação confidencial deste ambiente “se perder” em algum momento na rede? Provavelmente ela não deve ter saído de sua rota sozinha.

E se essa informação fosse desviada por alguma pessoa com a intenção de usá-la contra a empresa e isto acarretasse na perda de milhões, ou até bilhões? O que utilizar e de que formas se proteger contra este tipo de ataque?

Não há como afirmar que existe uma rede segura, nem que o protocolo de segurança X é totalmente confiável. Então, como posso enviar os dados sem que “caiam em mãos erradas”?

Não há como garantir que a informação não será interceptada, mas existem formas de tentar inibir os ataques, ou então, se por um acaso algum intruso consiga interceptar a informação, não a obtenha com clareza.

Não há nenhum sistema produzido até hoje que consiga garantir 100% de segurança em todo o momento. Algumas precauções podem ser tomadas e investimentos em segurança da informação devem ser realizados para garantir a integridade, a confiabilidade, o sigilo e a autenticidade da informação que trafega na rede. Hoje a forma mais eficaz de se proteger as informações é a criptografia, como será descrito nos capítulos seguintes.

2.1.2 Internet

A ***Internet*** e suas tecnologias avançam quase que diariamente. Isso remete a algumas questões, sendo uma das principais, se não a principal, a segurança na transmissão da informação pelos canais existentes. A ***Internet*** é utilizada hoje para todas as transações financeiras existentes, além de ser o principal meio para troca de informações entre empresas, pessoas, governos (em todas as suas esferas de poder).

A expansão do mundo digital em que vivemos, onde diversos aparelhos (computadores, celulares, ***tablets***, televisores, etc.) estão conectados à rede mundial de computadores, traz consigo desenvolvimento e mais interação para com as pessoas do mundo inteiro, mais isso traz como consequência crimes envolvendo esses meios. São milhares de pessoas, empresas e organizações sofrendo os mais diversos tipos de ataques, como podemos observar diariamente nos jornais.

O gráfico abaixo mostra um comparativo da quantidade de incidentes registrados em cada ano (1999 a 2014) pelo CERT.br referente aos ataques à segurança em serviços prestados na internet no País.

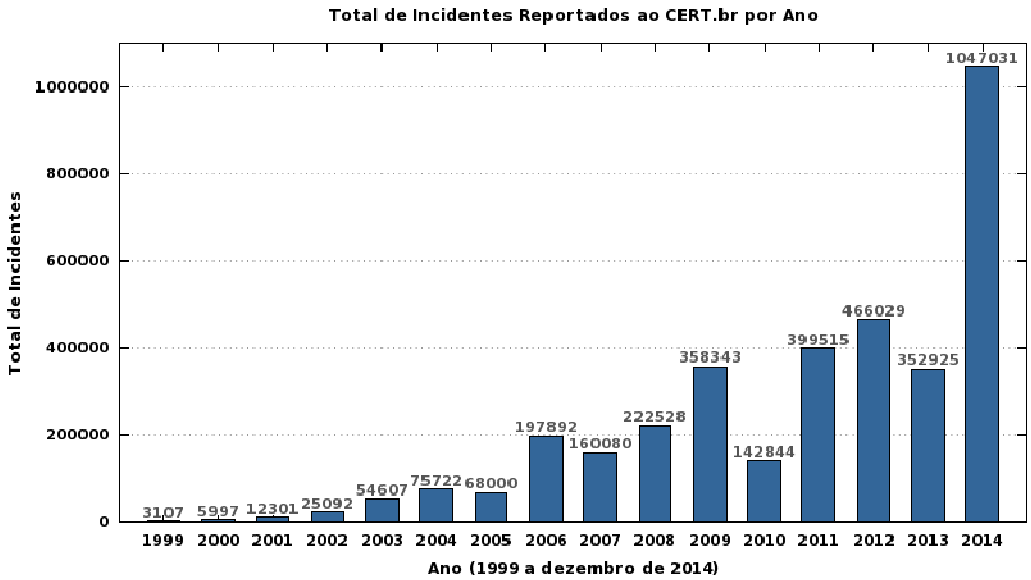

Gráfico 1 - Total de incidentes de segurança por ano reportados no Brasil
Fonte: CERT.br (2015)

O aumento considerável nos incidentes, como pode ser visto no gráfico 1, mostra o quanto é importante os investimentos com segurança. Se os atacantes conseguem entrar em um servidor e "atacá-lo" de diversas formas, facilmente eles conseguirão os dados sigilosos que lá estão armazenados ou por lá trafegam. Para que isso não aconteça deve-se seguir um protocolo de segurança bem definido.

> O surgimento do conjunto de protocolos ***Transmission Control Protocol/Internet Protocol*** (TCP/IP) e o advento da ***internet*** fizeram com que o alcance das invasões crescesse em proporções mundiais, uma vez que qualquer um pode atacar qualquer alvo (NAKAMURA E GEUS, 2007, p. 49).

Não há um meio de controlar os ataques, tendo em vista que qualquer pessoa pode acessar a ***Internet***, se registrar com um nome falso, inserir informações falsas e cometer crimes. E isso complica ainda mais quando utilizam de artifícios como mudança de endereço IP, clone de perfis e utilização de ***softwares*** que implicam na dificuldade de encontrar o verdadeiro autor do crime.

"A ***Internet*** deve ser considerada um ambiente hostil e, portanto, não confiável. Assim, todos os seus usuários devem ser considerados não confiáveis e potenciais atacantes." (NAKAMURA E GEUS, 2007, p. 59).

A informação na ***Internet*** está disponível para qualquer pessoa a qualquer momento, sendo esta, bem ou mal intencionada. Como hoje em dia é quase impossível

que uma organização sobreviva sem estar conectada com a ***Internet***, e por este meio tem de ser trocadas informações sigilosas, uma das formas de assegurar que estes dados não vão ser utilizados pelas pessoas erradas é criptografá-los. O que será discutido nos capítulos a seguir.

2.2 CRIPTOGRAFIA

Criptografia, originada do grego ***kriptos***, que significa “oculto, secreto, obscuro, ininteligível” e ***grapho***, que significa “escrita”, como o próprio nome já indica, é a forma de se ocultar, esconder, deixar ininteligível alguma informação ou dado.

As principais propriedades para que se confie na informação que está recebendo são: a integridade, a autenticidade, o não-repudio e o sigilo. A criptografia consegue implementar três delas, dando a possibilidade de se proteger e confiar em tal informação.

Além disso, para se alcançar as quatro propriedades utiliza-se a junção da criptografia com alguma outra tecnologia. Diversas são as formas encontradas para se criptografar, e para os mais diversos fins. Na história antiga é possível encontrar vários relatos de criptografia, desde o Egito Antigo até os dias de hoje.

Logo nos primórdios da civilização, quando os homens começaram a valorizar a moeda – esta que difere ricos e pobres e que traz comodidade e sensação de realização pessoal – e criaram grupos para alcançar seus objetivos, fez-se necessário codificar mensagens (para que os opositores não soubessem qual decisão tinha sido tomada e não pudessem se preparar) a fim de conseguir a primeira posição na corrida ao poder.

Uma das formas mais simples de se criptografar, ocorrida antes mesmo do surgimento da ***Internet*** foi empregada por Júlio César, onde mensageiros levavam as ordens dadas pelo comandante a suas tropas. Se a tropa inimiga interceptasse o mensageiro, a mesma não saberia o que estava escrito.

A forma de “enganar” os opositores, encontrada por Júlio César era o deslocamento de cada letra três posições no alfabeto, ou seja, se a mensagem fosse:

“ATACAR AO AMANHECER” estaria escrito “DXDFDU DR DPDQKHFHU”. Como a tropa de Júlio conhecia a “chave” ficava fácil decifrar a mensagem.

Fazendo uma analogia às redes de computadores existentes hoje, Júlio César seria o ponto inicial da mensagem (emissor), o deslocamento das três letras seria o algoritmo conhecido pelo emitente e pelo receptor – o qual possibilitava a encriptação e a decriptação da mensagem – o mensageiro seria o canal de transmissão da mensagem, a tropa de Júlio seria o ponto final da comunicação (receptor) e as tropas inimigas seriam os possíveis atacantes.

Este método é utilizado até hoje em alguns algoritmos. Conhecida como Cifra de César, tem variações no deslocamento, podendo ser empregado qualquer valor de “salto” entre as letras do alfabeto.

As tecnologias foram se desenvolvendo com o tempo, e o que difere o passado do presente é a quantidade de informações a serem compartilhadas, a forma com que são camufladas e a quantidade de tempo que a comunicação pode levar.

Hoje a criptografia não é utilizada tão somente para que os concorrentes não saibam quais as decisões as empresas e o governo tomam, mais sim, como uma das mais eficientes (mas mesmo assim, ainda falha) formas de se inibir o roubo e/ou adulteração de informações valiosas, como números e senhas de contas bancárias, informações privilegiadas de empresas, organizações e pessoas físicas.

> De fato, no mundo atual, onde a comunicação está cada vez mais onipresente na vida das pessoas, a proteção de toda essa comunicação deve ser garantida, bem como a privacidade dos usuários. Dessa maneira, a criptografia já é usada em muitas soluções do dia-a-dia dos usuários de todos os níveis (NAKAMURA E GEUS, 2007, p. 302).

Quanto mais a criptografia na era computadorizada avança e fica aparentemente mais segura, maiores são as tentativas dos criminosos de alcançar uma forma de decifrá-las. Cria-se assim um círculo vicioso, pois geralmente essas tentativas são bem sucedidas e novamente tem-se o dever de encontrar novas formas de inibir tais ataques.

A segurança de um protocolo de criptografia está concentrada na chave secreta utilizada. Caso essa chave seja roubada toda a segurança da comunicação estará perdida. Existem várias formas de criptografia. Neste trabalho serão abordadas a

simétrica, ou mais conhecida como chave privada e a assimétrica, também conhecida como chave pública.

A forma mais segura encontrada hoje é a criptografia quântica, mas ainda é pouco utilizada, pois necessita de um poder computacional muito grande e um vasto investimento, sendo assim, fica inviável implementar este método para pequenas quantidades de informação.

2.3 CHAVES SIMÉTRICAS E ASSIMÉTRICAS

A criptografia se resume em: o emissor X criptografa o texto por meio de um algoritmo para que o conteúdo seja camuflado. O receptor Y decifra a mensagem por meio do algoritmo de deciframento correspondente e encontra o conteúdo original.

Caso algum intruso conhecesse de alguma forma o algoritmo, ele conseguiria a informação com tanta facilidade quanto o receptor Y. Para tentar inibir esse tipo de ataque utiliza-se então, na criptografia simétrica, chaves privadas, onde o emissor X e o receptor Y as conhecem previamente, e somente com elas será possível cifrar e decifrar as mensagens.

> "Existe o problema da necessidade de distribuição das chaves secretas a serem utilizadas pelos usuários, que deve ser feita de maneira segura. O problema está na dificuldade de enviar a chave gerada para o usuário, pois o canal de comunicação ainda não é seguro." (NAKAMURA E GEUS, 2007, p. 303).

A segurança desse tipo de criptografia não se encontra então no algoritmo que camufla a mensagem e sim na chave secreta, e esta sim deverá ser mantida em sigilo tanto pelo emissor X quanto pelo receptor Y. Este é um método simples, e é exatamente esta a falha dele. Se algum intruso conseguir a chave ficará fácil conseguir o conteúdo da mensagem.

> O principal problema residente na utilização deste sistema de criptografia é que quando a chave de ciframento é a mesma utilizada para deciframento, ou esta última pode facilmente ser obtida a partir do conhecimento da primeira, ambas precisam ser compartilhadas previamente entre origem e destino, antes de se estabelecer o canal criptográfico desejado, e durante o processo de compartilhamento a senha pode ser interceptada, por isso é fundamental utilizar um canal seguro durante o compartilhamento, este independente do destinado

> à comunicação sigilosa, uma vez que qualquer um que tenha acesso à senha poderá descobrir o conteúdo secreto da mensagem (OLIVEIRA, 2012, p. 02).

O compartilhamento da chave passa a ser o maior problema do método. Não há como dizer que existe um canal em que a chave possa ser transportada totalmente segura, pois se existisse, não seria enviada a chave e sim a mensagem. A figura abaixo demonstra o esquema de criptografia por chave simétrica

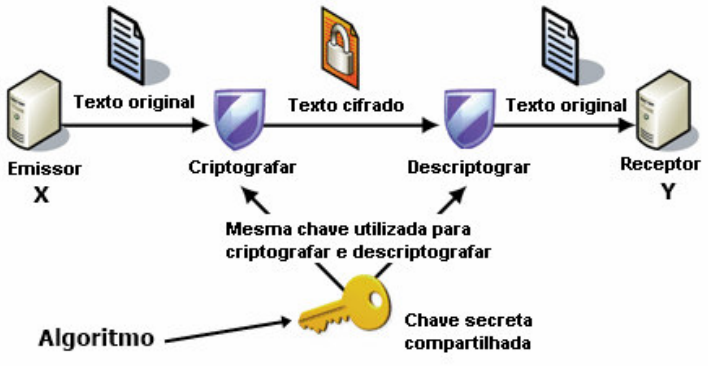

Figura 1 - Modelo de encriptação por chave simétrica
Fonte: Adaptado de 4Java (2015)

Além disso, não há como utilizar uma única chave secreta para comunicação entre todos os participantes. Quando um terceiro descobre a chave secreta, facilmente obterá o conteúdo da mensagem, então faz-se necessária a criação de uma chave secreta para cada comunicação ponto-a-ponto.

> Seja o número de participantes igual a N, o número de chaves que cada participante deve gerenciar é igual a N-1. A gerência deste número elevado de chaves é um problema que deve ser resolvido quando se usam algoritmos simétricos (BRAZIL, 2007, p. 19). Outro problema é o uso de chaves secretas diferentes para cada tipo de comunicação e também para cada mensagem, o que faz com que seu gerenciamento se torne muito complexo. (NAKAMURA E GEUS, 2007, p. 303).

Já a criptografia por chave assimétrica apresenta uma melhora, e talvez a solução para o problema da chave compartilhada encontrada na criptografia simétrica. Neste modelo, existem duas chaves, uma chamada pública e a outra chamada privada.

A chave pública é conhecida por qualquer pessoa que deseje se comunicar com os demais de modo seguro, já a chave privada é conhecida somente pelo seu

respectivo titular. É com a chave privada que o titular conseguirá decifrar uma mensagem encaminhada para ele por meio de sua chave pública.

Nas figuras abaixo representa-se o modelo de chave assimétrica, onde a chave verde representa a chave pública e a chave rosa representa a chave privada:

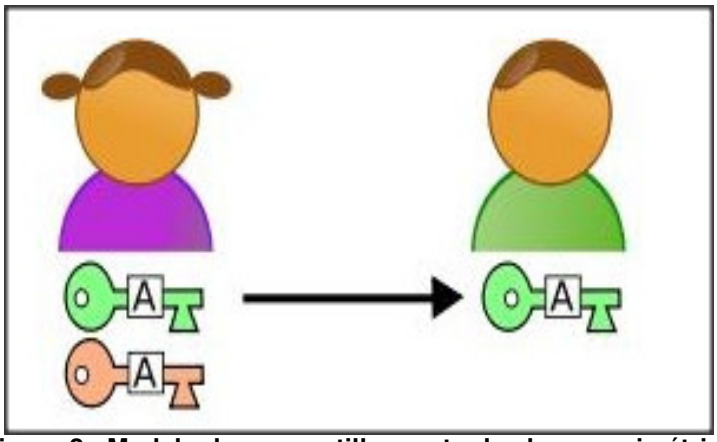

Figura 2 - Modelo de compartilhamento da chave assimétrica
Fonte: Castelló e Vaz

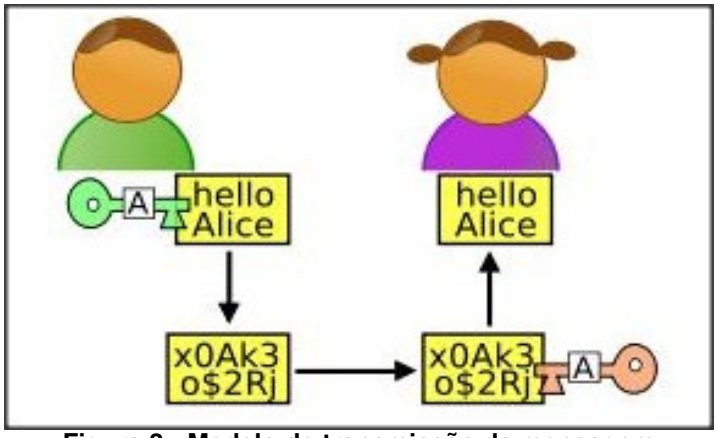

Figura 3 - Modelo de transmissão da mensagem
Fonte: Castelló e Vaz

A parte importante deste método é que não há o compartilhamento de chaves privadas pelos canais de comunicação, mas, o ponto fraco desse método se encontra no fato de que ele necessita de um tempo de processamento maior do que na criptografia simétrica, o que pode limitar a utilização em alguns casos.

“O algoritmo assimétrico minimiza o problema de troca de chaves, pois não é necessário um canal seguro para tal. Porém, ele é cerca de 60 a 70 vezes mais lento que os algoritmos simétricos.” (NAKAMURA e GEUS, 2007, p. 304).

Qualquer um pode possuir a chave pública de um determinado ponto e mandar uma mensagem criptografada para o dono da chave secreta, sendo assim, este só poderá ler a mensagem utilizando sua chave secreta. Neste modelo a chave pública não poderá ser obtida com o conhecimento da chave privada e vice-versa.

> Para entender o conceito, basta pensar num cadeado comum protegendo um determinado bem. A mensagem é este bem, e o cadeado, que pode ficar exposto, é a chave pública. Apenas quem tiver uma chave particular (privada) que consiga abrir o cadeado poderá acessar a mensagem. A principal vantagem deste método é a sua segurança, pois não é preciso (nem se deve) compartilhar a chave privada. Por outro lado, o tempo de processamento de mensagens com criptografia assimétrica é muitas vezes maior do que com criptografia simétrica, o que pode limitar seu uso em determinadas situações (OLIVEIRA, 2012, p. 4).

Os problemas que a chave pública apresenta são: a lentidão quando há uma grande quantidade de informações a serem criptografadas, a incerteza de que a chave pública é realmente a chave do usuário X, entre outros.

2.4 ALGORITMOS DE CRIPTOGRAFIA

Abaixo elenca-se os principais algoritmos para criptografia tanto simétrica como assimétrica. São códigos de computador que misturam lógica de programação, lógica matemática e protocolos para tornar mensagens claras em textos inelegíveis e possibilitar a transferência desses dados por meio de canais de comunicação (rede) tanto internos quanto externos.

2.4.1 Algoritmos Para Criptografia Simétrica

“Os algoritmos de chave simétrica têm como característica a rapidez na execução, porém eles não permitem a assinatura e a certificação digitais.” (NAKAMURA

E GEUS, 2007, p. 303). Criptografar simetricamente uma mensagem tende a ser simples, se comparado com outros métodos.

Toda a segurança está concentrada no sigilo das chaves, o que complica o sistema, pois elas devem ser compartilhadas ponto-a-ponto, e devem ser transmitidas em uma rede segura para que alguém não autorizado não consiga acessá-la, tendo em vista que a maioria dos algoritmos de criptografia são abertos para estudo.

A seguir descreve-se alguns dos algoritmos mais conhecidos e utilizados que implementam a criptografia simétrica:

Algoritmo RC e suas versões RC2, RC4, RC5, RC6. São basicamente uma sucessão de algoritmos que apresentam melhoras a cada versão. Por conseguirem alcançar uma segurança maior do que outros algoritmos simétricos são bastante utilizados.

A particularidade deles são as chaves de tamanho variável, possibilitando criptografar grandes quantidades de dados em menor tempo se comparados a outros algoritmos de criptografia simétrica.

O RC2 é um algoritmo proprietário, restrito aos implementadores licenciados pela RSA Data Security Inc. Ele não é um algoritmo iterativo, obtém como entrada blocos de 64 ***bits*** (8 ***bytes***) e as chaves variam de tamanho. O mais comum de se utilizar neste algoritmo são as chaves de 128 ***bits*** (16 ***bytes***).

O RC4 foge da arquitetura normal, não trabalha com blocos e sim com um fluxo contínuo de entradas e saídas de ***bytes*** cifrados ou decifrados. Este algoritmo se difere dos outros simétricos, sendo mais veloz em vários casos. Por ser a versão melhorada do anterior, ainda trabalha com as chaves de tamanho variável, sendo a mais utilizada a de 16 ***bytes***.

> O RC4 consiste de duas fases: a primeira, que se denota por KSA ("***Key-Scheduling Algorithm***"), que transforma uma chave K (cujo comprimento normalmente varia entre 5 e 32 ***bytes*** e é representado por l) numa permutação S do conjunto dos ***bytes*** {0, . . ., N − 1}, em que N = 256; a segunda, que se denota por PRGA ("***Pseudo-Random Generation Algorithm***"), que usa S para gerar uma sequência pseudoaleatória de ***bytes*** (FERREIRA, 2006, p. 06).

Usualmente, em outra técnica de criptografia, este algoritmo é combinado com algum outro assimétrico para possibilitar mais confiabilidade, segurança e agilidade.

Na versão RC5, o algoritmo volta a trabalhar com blocos e caracteriza-se por ser flexível, pois nele os blocos de entrada e o número de iterações do algoritmo devem ser pré-determinadas e a chave pode ter qualquer comprimento em um intervalo de valores determinados.

Este algoritmo é usualmente escrito RC5 w/r/b onde w é o número de palavras do bloco, r é o número de rodadas e b são os ***bytes*** da chave. Os parâmetros desse algoritmo devem ser ajustados conforme as necessidades de cada caso.

A mensagem de entrada é fornecida ao algoritmo em forma de dois blocos de tamanho definido pela particularidade do caso (tipicamente são utilizados tamanhos de 16, 32 ou 64 ***bytes***), como saída a mensagem camuflada tem tamanho e forma idêntica à entrada. O número de iterações pode ocorrer no intervalo de 0 a 255 e o número de ***bytes*** da chave pode ser escolhido também no intervalo de 0 a 255.

O algoritmo RC6 também trabalha com blocos, baseado no RC5, ele foi desenvolvido para uma competição realizada pelo NIST e foi um dos finalistas. Essa competição foi realizada para definir o algoritmo que substituiria o DES, sendo o vencedor nomeado de AES.

O algoritmo DES, é baseado em um algoritmo desenvolvido pela IBM, chamado Lúcifer. O objetivo deste é que seja muito difícil calcular a chave mesmo conhecendo o algoritmo, a mensagem cifrada e a mensagem original. O algoritmo é difícil de ser eficientemente implementado em ***software***, mas pode ser implementado em ***hardware*** e apresentar melhor eficiência.

O algoritmo “cifra blocos de 64 ***bits*** com uma chave de 56 ***bits*** (na verdade existem mais oito de paridade dando um total de 64 ***bits***) e uma função de embaralhamento chamada função Feistel.” (BRAZIL, 2007, p. 19).

Apesar de permitir cerca de setenta e dois quadrilhões de combinações possíveis, este algoritmo foi quebrado por força bruta em um desafio lançado na ***Internet*** no ano de 1997. A partir do ano de 1993 passou a ser recomendada a utilização do algoritmo 3DES, que nada mais é do que uma variação simples do ***Data Encryption Standard***.

O nome deste remete ao seu modo de funcionamento: ele se utiliza de três ciframentos sucessivos e chega a empregar duas ou três chaves diferentes na mesma

versão. Este algoritmo tem se mostrado seguro, mas seu ponto negativo é a lentidão, tornando inviável sua utilização.

O algoritmo IDEA, utiliza blocos fixos de 8 ***bytes*** e chaves de 16 ***bytes***. Realiza oito rodadas de iterações, alternando operações de três grupos algébricos de estruturas distintas e opera em palavras de 2 ***bytes***. Este algoritmo pode ser utilizado em qualquer modo encadeado. As oito rodadas em ***software*** fazem com que este algoritmo se torne duas vezes mais eficiente que o algoritmo DES.

> Foi criado em 1991 por James Massey e Xuejia Lai e possui patente da suíça ASCOM Systec. O algoritmo é estruturado seguindo as mesmas linhas gerais do DES. Mas na maioria dos microprocessadores, uma implementação por ***software*** do IDEA é mais rápida do que uma implementação por ***software*** do DES. O IDEA é utilizado principalmente no mercado financeiro e no PGP, o programa para criptografia de ***e-mail*** pessoal mais disseminado no mundo (OLIVEIRA, 2012, p. 03).

O algoritmo AES, utiliza o mesmo modelo de criptografia de blocos. “O AES é baseado no algoritmo Rijndael, inventado por Joan Daeman e Vincent Rijmen. O algoritmo Rijndael permite a escolha do tamanho das chaves e dos blocos (separadamente).” (FERREIRA, 2006, p. 42).

Uma particularidade que o AES não herdou do Rijndael é a possibilidade da escolha do tamanho dos blocos, ele fixa o tamanho em 128 ***bits***.

> O Rijndael é um algoritmo de criptografia de blocos, trabalhando com blocos de 128 ***bits*** e chaves de 128, 192 ou 256 ***bits***. O Rijndael original foi desenvolvido para suportar tamanhos diferentes de blocos de dados e de chaves, porém, estes não são adotados na versão AES (TREVISAN ET AL., 2013, p. 15).

Neste algoritmo o número de rodadas depende do tamanho da chave existente, por exemplo, se a chave for de tamanho quatro haverá dez rodadas, se a chave for de tamanho seis haverá doze rodadas, se a chave for de tamanho oito haverá quatorze rodadas. Existe uma chave principal e a partir dela são geradas uma chave para cada rodada e cada uma delas é agrupada da mesma maneira que o bloco de dados.

> O ***Advanced Encryption Standard*** (AES) é uma cifra de bloco, anunciado pelo ***National Institute of Standards and Technology*** (NIST) em 2003, fruto de concurso para escolha de um novo algoritmo de chave simétrica para proteger informações do governo federal, sendo adotado como padrão pelo governo dos Estados Unidos, é um dos algoritmos mais populares, desde 2006, usado para criptografia de chave simétrica, sendo considerado como o padrão substituto do DES. O AES tem um tamanho de bloco fixo em 128 ***bits*** e uma chave com

> tamanho de 128, 192 ou 256 ***bits***, ele é rápido tanto em ***software*** quanto em ***hardware***, é relativamente fácil de executar e requer pouca memória (OLIVEIRA, 2012, p. 03).

O algoritmo Blowfish foi desenvolvido por Bruce Schneier, que deixou o código livre de licenças e aberto para quem quiser utilizá-lo. O autor diz que "a criptografia é uma ciência que deve estar a disposição de todos". Como todos os demais algoritmos simétricos, é uma cifra de blocos, com um bloco de tamanho fixo de 64 ***bits*** e uma chave que varia de 32 a 448 ***bits***.

"O Blowfish ganhou uma grande aceitação no mercado sendo utilizado em diversas aplicações, dentre elas, o ***Nautilus*** e o ***PGPfone***. O Blowfish utiliza uma função não inversível, caixas-S (***S-boxes***) e uma rede de Feistel. Este algoritmo nunca foi quebrado." (MORAES, 2004, p. 82).

Este algoritmo oferece a escolha de melhor desempenho ou melhor segurança dependendo do tamanho da chave a ser utilizada. Ele pode ser usado como substituto do DES e do IDEA, podendo ser implementado tanto em aplicações domésticas, quanto comerciais. O Blowfish foi desenvolvido para ser rápido, compacto, simples e seguro.

Por padrão os algoritmos que possuem um intervalo de valores possíveis deveriam utilizam chaves do mesmo tamanho que os blocos, mas, na prática isto pouco acontece. Quando maior a chave utilizada, mas difícil será aplicar um ataque de força bruta para conseguir a chave e decifrar o texto da mensagem.

A tabela abaixo demonstra os tamanhos das chaves e os tamanhos dos blocos aceitos em cada algoritmo de criptografia simétrica.

Tabela 1 - Tamanho de chaves e blocos utilizados nos algoritmos simétricos

Algoritmo	Tamanho das Chaves	Tamanho dos Blocos
RC2	8 a 1024 ***bits***	64 ***bits***
RC4	1 a 256 ***bits***	-
RC5	1 a 2040 ***bits***	32-64-128 ***bits***
RC6	1 a 255 ***bits***	32 ***bits***
DES	56 ***bits***	64 ***bits***
3DES	3 chaves de 56 ***bits***	64 ***bits***
IDEA	128 ***bits***	64 ***bits***
AES	128-192-256 ***bits***	128 ***bits***
Blowfish	32 a 448 ***bits***	64 ***bits***

2.4.2 Algoritmos Para Criptografia Assimétrica

A criptografia assimétrica, assim como a simétrica necessita manter segura a chave privada. A diferença é que neste modelo são utilizadas duas chaves, uma visível a todos que querem se comunicar com um determinado ponto e outra que deve ser mantida em segredo.

“Nesse método, uma pessoa deve criar uma chave de codificação e enviá-la a quem for lhe mandar informações. Essa é a chave pública. Outra chave deve ser criada para a decodificação. Esta, a chave privada, é secreta.” (SOUZA E SILVA, 2013, p. 03).

Um algoritmo para implementar a criptografia de chave assimétrica necessita considerar alguns pontos: deve-se conseguir criptografar e decriptografar a mensagem dada a chave correspondente; deve ser computacionalmente inviável obter a chave privada a partir da chave pública e deve ser computacionalmente inviável conseguir a chave privada por meio de ataques de força bruta.

“Os algoritmos de chave pública ou assimétrica, como RSA, Rabin e outros, podem possibilitar, além do sigilo, integridade, não-repúdio e autenticidade. É possível ainda que a assinatura e a certificação digitais possam ser utilizadas.” (NAKAMURA E GEUS, 2007, p. 303).

Abaixo elenca-se as características dos principais e mais utilizados algoritmos de criptografia de chave pública.

O algoritmo RSA é atualmente o mais utilizado e até o momento o mais poderoso dos algoritmos que criptografam por meio de chave pública. A chave pública é obtida pela fatoração de dois números grandes primos e a chave privada é a multiplicação destes mesmos números.

> O RSA utiliza números primos. A premissa por trás do RSA consiste na facilidade de multiplicar dois números primos para obter um terceiro número, mas muito difícil de recuperar os dois primos a partir daquele terceiro número. Isto é conhecido como fatoração. Por exemplo, os fatores primos de 3.337 são 47 e 71. Gerar a chave pública envolve multiplicar dois primos grandes; qualquer um pode fazer isto. Derivar a chave privada a partir da chave pública envolve fatorar um grande número. Se o número for grande o suficiente e bem escolhido, então ninguém pode fazer isto em uma quantidade de tempo razoável. Assim, a segurança do RSA baseia se na dificuldade de fatoração de números grandes. Deste modo, a fatoração representa um limite superior do tempo necessário para quebrar o algoritmo. Uma chave RSA de 512 ***bits*** foi quebrada em 1999 pelo Instituto Nacional de Pesquisa da Holanda, com o apoio de cientistas de mais 6 países. Levou cerca de 7 meses e foram utilizadas 300 estações de trabalho para a quebra. No Brasil, o RSA é utilizado

> pela ICP-Brasil, no seu sistema de emissão de certificados digitais, e a partir do dia 1º de janeiro de 2012, as chaves utilizadas pelas autoridades certificadoras do país, passam a serem emitidas com o comprimento de 4.096 ***bits***, em vez dos 2.048 ***bits*** atuais (OLIVEIRA, 2012, p. 04).

O algoritmo ElGamal se apresenta seguro por utilizar-se de um problema matemático chamado de "logaritmo discreto" em um corpo finito, envolvendo a manipulação matemática de grandes quantidades numéricas. Este algoritmo é frequentemente utilizado em assinaturas digitais.

> O algoritmo envolve a manipulação matemática de grandes quantidades numéricas. Sua segurança advém de algo denominado problema do logaritmo discreto. Assim, o ElGamal obtém sua segurança da dificuldade de calcular logaritmos discretos em um corpo finito, o que lembra bastante o problema da fatoração (OLIVEIRA, 2012, p. 05).

O algoritmo Rabin se baseia na dificuldade computacional de uma pessoa não autorizada, calcular o texto legível, a partir do conhecimento do texto ilegível correspondente. Este algoritmo utiliza do problema matemático de se extrair raiz quadrada em anéis.

> Calcula-se dois números inteiros primos e longos (i. e., com centenas de ***bits***) chamados *q* e *r*; e calcula-se o seu produto $n = q.r$. Onde o comprimento de *q* seja próximo de *r* tornando inviável a fatoração rápida de n em primos. A chave secreta $S = (q, r)$ é guardada com cuidado e a pública $P = (n)$ é enviada para qualquer pessoa amiga (SOUSA, 2005, p. 06).

A segurança deste algoritmo também se equivale a dificuldade de se fatorizar um número inteiro em primos.

Algoritmos criptográficos modificados para basear-se em curvas elípticas podem garantir o mesmo nível de segurança do RSA, utilizando-se de chaves com o tamanho bem menor.

> Em 1985, Neal Koblitz e V. S. Miller propuseram de forma independente a utilização de curvas elípticas para sistemas criptográficos de chave pública. Eles não chegaram a inventar um novo algoritmo criptográfico com curvas elípticas sobre corpos finitos, mas implementaram algoritmos de chave pública já existentes, como o algoritmo de Diffie-Hellman, usando curvas elípticas. Assim, os sistemas criptográficos de curvas elípticas consistem em modificações de outros sistemas (o ElGamal, por exemplo), que passam a trabalhar no domínio das curvas elípticas, em vez de trabalharem no domínio dos corpos finitos. Eles possuem o potencial de proverem sistemas criptográficos de chave pública mais seguros, com chaves de menor tamanho (OLIVEIRA, 2012, p. 05).

2.5 LINGUAGEM DE PROGRAMAÇÃO – JAVA

Java é uma linguagem de programação desenvolvida pela SUN Microsystems (adquirida em 2010 pela Oracle) relativamente nova se comparada com outras linguagens, seu lançamento foi no dia 23 de maio de 1995.

O intuito do Java é manter o poder computacional do C++, introduzindo funcionalidades que proporcionassem segurança, robustez e portabilidade – sendo esta última, uma de suas maiores vantagens, a linguagem funciona da mesma forma em diversos sistemas operacionais.

Sua arquitetura tem a característica de trabalhar com uma máquina virtual. Também gerencia a alocação/liberação da memória por meio do ***Garbage Collection*** e possui um módulo de garantia de segurança do código (é quase impossível criar um vírus nesta linguagem).

Outra característica notável da linguagem é sua capacidade de integração com objetos distribuídos, com a grande maioria dos sistemas de gerenciamento de banco de dados e até mesmo com outras linguagens de programação.

Não há como falar de Java sem remeter-se ao fato de que esta é a linguagem que popularizou a orientação à objetos, que já era conhecida desde os anos 70, mas foi amplamente difundida somente nos anos 90. Este paradigma tenta adotar formas mais próximas do mecanismo humano para gerenciar a complexidade dos sistemas.

“Nesse paradigma, o mundo real é visto como sendo constituído de objetos autônomos, concorrentes, que interagem entre si, e cada objeto tem seu próprio estado (atributos) e comportamento (métodos), semelhante a seu correspondente no mundo real.” (MENDES, 2009, p. 18).

Para agilizar o processo do desenvolvimento de um ***software***, Java possui mais uma característica – suas API’s, que são um conjunto de classes previamente implementadas, disponibilizadas para ajudar os programadores.

> Uma das vantagens de utilizar a linguagem de programação Java é a sua infinidade de API´s desenvolvidas especialmente para fins específicos. Muito se vê, navegando pela ***internet***, vários algoritmos de criptografia prontos, criados

> por programadores que adaptaram até mesmo de outras linguagens a forma de criar dados cifrados e de decifrar os mesmos. O problema é a necessidade de grande conhecimento pra lidar com esses tipos de algoritmos, lidar com deslocamento de ***bytes***, se preocupar com a segurança do algoritmo, fazendo com que seja de difícil descoberta, não deixando que dados sejam decifrados (PAULINO, 2009, p. 27).

Como descrito acima, para implementar todos os métodos de um algoritmo de criptografia, o programador deve conhecer minuciosamente as funções matemáticas e lógicas, bem como as questões de segurança que ele possui – para que o mesmo não perca suas características originais e sua utilidade.

Então para auxiliar neste desenvolvimento também é possível utilizar-se de uma API. “A linguagem Java fornece uma API poderosa para se trabalhar com criptografia de dados.” (LAZANHA, 2005, p. 56). Esta citação se refere a JCA.

Esta arquitetura fornece vários serviços de criptografia de dados. O programador pode escolher entre utilizar os métodos prontos ou melhorar sua implementação. Com a JCA é possível criptografar os dados informando somente o algoritmo que deseja utilizar, a chave gerada e o conjunto de dados puros.

Não é necessário portanto aprofundar-se nas questões lógicas/matemáticas do algoritmo para poder utilizá-lo com segurança. A Oracle também fornece a JCE, que é uma extensão da JCA.

Esta possui funcionalidades mais especificas de criptografia, gerando chaves secretas, códigos de autenticação e algoritmos de ***digest***, entre outras. Ela também permite uma iteração com dispositivos criptográficos como ***tokens***, leitoras de ***Smart Card***, etc.

Uma das funcionalidades da JCE é denominada ***Cipher*** (que será utilizada neste trabalho), a qual é implementada por meio do pacote javax.cypto. Este disponibiliza métodos que geram a cifra para criptografar/decriptografar, conforme a especificação de cada algoritmo.

> Outro fator referente à linguagem Java é o de permitir implementar aplicativos baseados em ***Intranet*** e ***Internet*** e qualquer outro ***software*** para dispositivos que se comuniquem em uma rede, fato que só vem consolidar a motivação e importância da implementação do algoritmo criptográfico em Java, obtendo assim facilidade de implementações de aplicativos que se comunicam em rede juntamente com a segurança (BUGATTI, 2005, p. 66).

É importante frisar que não são todos os algoritmos de criptografia que a JCA e a JCE implementam. Segundo a documentação da Oracle para o Java 8, os algoritmos que a classe ***Cipher*** implementa são: AES, AESWrap, ARCFOUR, Blowfish, DES, DESede (nominação do 3DES), DESedeWrap, ECIES, PBEWithMD5AndDES, PBEWithHmacSHA1AndDESede, RC2, RC4, RC5 e RSA. Os métodos padrões implementados pela ***Cipher***, são descritos no quadro 1.

Método	Descrição
doFinal()	Conclui o processo de criptografia/decriptografia.
getAlgorithm()	Retorna o nome do algoritmo que gerou a cifra em questão.
getBlockSize()	Retorna o tamanho do bloco (em ***bytes***).
getExemptionMechanism()	Retorna um objeto contendo o mecanismo de isenção utilizado com a cifra.
getInstance()	Retorna a cifra do algoritmo desejado – utilizada para fazer as transformações.
getIV()	Retorna o vetor de inicialização utilizado.
getMaxAllowedKeyLength()	Retorna o tamanho máximo da chave (em ***bits***) que pode ser utilizada, de acordo com a descrição do algoritmo e conforme a política de segurança do JCE instalada.
getMaxAllowedParameterSpec()	Retorna um objeto ***AlgorithmParameterSpec*** que contém o valor do parâmetro máximo da cifra de acordo com a política de segurança do JCE instalada.
getOutputSize()	Retorna o tamanho do ***buffer*** de saída requerido (em ***bytes***).
getParameters()	Retorna os parâmetros utilizados na cifra.
getProvider()	Retorna o nome do fornecedor do objeto ***Cipher*** em questão.
Init()	Inicializa a cifra com a chave secreta.
Unwrap()	"Desembrulha" a chave que está encoberta.
Update()	Faz a criptografia/decriptografia de vários blocos simultâneos.
updateAAD()	Faz a criptografia/decriptografia de multi-partes com a autenticação de dados adicionais (AAD).
Wrap()	"Embrulha" a chave.

Quadro 1 - Métodos implementados pela classe *Cipher*
Fonte: Documentação *Oracle* – Java 8

Para que seja possível criptografar/decriptografar um dado é necessário possuir uma ou duas chave(s) e em alguns casos o algoritmo exige um vetor de inicialização

(aumentando a segurança do processo). As classes elencadas abaixo foram utilizadas neste trabalho, tanto na criptografia simétrica quanto na assimétrica.

Os métodos demonstrados abaixo são responsáveis pela criação e gerenciamento da(s) chave(s) e do vetor. Cada classe descrita no quadro 2 se transforma em um objeto a ser utilizado pela classe criada pelo programador.

Classe	Método	Descrição
KeyGenerator	generateKey()	Gera a chave secreta.
KeyGenerator	getAlgorithm()	Retorna o nome do algoritmo que gerou a chave.
KeyGenerator	getInstance()	Retorna um objeto ***KeyGenerator*** que gera a chave a partir do algoritmo especificado.
KeyGenerator	getProvider()	Retorna o nome do fornecedor do ***KeyGenerator***.
KeyGenerator	init()	Inicializa o gerador da chave a partir dos parâmetros especificados.
SecretKey	Não possui	É uma interface, seu único propósito é agrupar e fornecer a segurança para as chaves secretas.
IvParameterSpec	getIV()	Esta classe gera o vetor de inicialização e o método retorna o vetor.
KeyPairGenerator	generateKeyPair()	Gera o par de chaves (pública e privada).
KeyPairGenerator	genKeyPair()	Gera o par de chaves (pública e privada) – equivale ao método ***generateKeyPair()***.
KeyPairGenerator	getAlgorithm()	Retorna o nome do algoritmo que gerou o par de chaves.
KeyPairGenerator	getInstance()	Retorna um objeto ***KeyPairGenerator*** que gera o par de chaves a partir do algoritmo especificado.
KeyPairGenerator	getProvider()	Retorna do nome do fornecedor do ***KeyPairGenerator***.
KeyPairGenerator	initialize()	Inicializa o gerador do par de chaves de acordo com os parâmetros especificados.

Quadro 2 - Descrição das classes e dos métodos de segurança
Fonte: Documentação *Oracle* – Java 8

Segundo a documentação da Oracle, os algoritmos que as classes ***KeyGenerator*** e ***SecretKey*** implementam são: AES, ARCFOUR (nominação do RC4), Blowfish, DES, DESede, HmacMD5, HmacSHA1, HmacSHA224, HmacSHA256, HmacSHA384, HmacSHA512 e RC2.

Nota-se que o algoritmo RC5 utilizado neste trabalho não tem sua implementação nestas duas classes, para isso será necessária a utilização de outra API de criptografia em conjunto com a JCA, como será descrito no capítulo 2.6.4

Já a classe ***KeyPairGenerator*** implementa os algoritmos assimétricos: DiffieHellman, DSA, RSA e EC.

Tendo em vista que cada jurisdição internacional tem um padrão de segurança definido, a JCA implementa o padrão da criação das chaves com o menor tamanho possível, de acordo com a especificação de cada algoritmo.

Para utilizar tamanhos de chaves superiores deve ser implementado o método ***init()*** ou o método ***initialize()***. Mas, para que os mesmos suportem tamanhos de chaves variáveis é necessário instalar o pacote "***Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files***" especifico para a versão do Java utilizada, disponível no site da Oracle.

2.6 IMPLEMENTAÇÃO DOS ALGORITMOS EM JAVA

O objetivo deste trabalho não é aprofundar-se nas funções especificas de cada algoritmo e sim testar o tempo e o desempenho deles. Tendo em vista que a linguagem de programação Java possui métodos que realizam a criação das chaves, criptografam e descriptografam a mensagem, não será implementado o passo a passo para tal, mas sim, serão utilizados os métodos da API de criptografia.

"A maioria dos algoritmos criptográficos geralmente se utilizam de estruturas e funções criptográficas semelhantes. Tais estruturas podem ser "levemente" modificadas ou possuírem outra denominação mas o conceito permanece basicamente o mesmo." (LAZANHA, 2005, p. 40).

Tomando como base esta descrição é possível entender porque a API de criptografia implementa vários algoritmos – cada um com sua particularidade – e porque ela pode ser utilizada para os testes.

Abaixo são descritos os algoritmos de criptografia implementados neste trabalho: AES, Blowfish, RC2, RC5, RC4, RSA, como eles funcionam e como os métodos pré definidos são implementados para cada um deles.

2.6.1 Advanced Encryption Standard

Algoritmos de criptografia são um conjunto de funções matemáticas em que os dados são submetidos para que possam ser escondidos de forma eficaz.

No caso do algoritmo Rijndael (no qual o AES baseia-se), são utilizadas funções de soma de ***bytes***, realização de XOR nos ***bits*** de cada ***byte***, ***S-Box***, deslocamento e multiplicação de ***bytes*** realizando redução modular com um polinômio irredutível, para chegar a versão criptografada da mensagem.

> Para o funcionamento do AES são necessários alguns dados como a ***S–Box*** (tabela de substituição estática), o estado (que é o bloco de dados de entrada sendo modificado pelas transformações do algoritmo), a chave, e a chave de expansão (uma versão modificada da chave). (TREVISAN ET AL., 2013, p. 15).

São realizadas quatro transformações durante a fase de cifragem: substituir os ***bytes*** do estado em ***bytes*** de ***S-Box***; realizar a rotação cíclica das linhas do estado, a segunda em 1 casa, a terceira em 2 casas e a quarta em 3 casas; transformar os dados das colunas do estado multiplicando por um polinômio irredutível fixado e concatenar as colunas com uma das chaves geradas na rotina de expansão.

Para decifrar a mensagem são implementadas quatro transformações proporcionalmente inversas às de cifragem. O algoritmo contendo o método de criptografar por meio do AES utilizado neste trabalho foi implementado por Higor Medeiros em seu artigo “Utilizando Criptografia Simétrica em Java”.

Como a versão original do código trabalha com criptografia de texto (***string***), foram feitas modificações para simplificar e atender aos quesitos do trabalho. Este código foi escolhido por utilizar métodos do pacote javax.crypto do Java, não sendo necessária a implementação do passo a passo para a criação da chave e a codificação/decodificação dos blocos.

No código implementado por Higor Medeiros, define-se uma chave de 16 ***bytes*** (128 ***bits***) estática, como pode ser visto na primeira linha do quadro abaixo. Cria-se então uma instância do algoritmo AES e transforma-se a ***string*** “chaveencriptacao” em uma chave secreta, conforme a especificação do algoritmo de criptografia.

Na última linha do quadro é possível ver a instância do algoritmo AES com o modo de cifra CBC e um PKCS5Padding, este irá completar os blocos se o tamanho da

informação a ser criptografada não for múltiplo de 16, com a implementação especificada pelo provedor SunJCE.

```
static String chaveencriptacao = "0123456789abcdef";
keygenerator = KeyGenerator.getInstance("AES");
chaveAES = new SecretKeySpec(chaveencriptacao.getBytes("UTF-8"), "AES");
cifraAES = Cipher.getInstance("AES/CBC/PKCS5Padding", "SunJCE");
```

Quadro 3 - Código para gerar a chave e a cifra algoritmo AES
Fonte: Medeiros (2014)

Como o interesse é em criar uma chave não estática para cada comunicação realizada, as alterações no código acima são apresentadas no quadro abaixo. Ao invés de utilizar o método ***SecretKeySpec()*** para criar uma chave secreta, utiliza-se o método ***generateKey()***. Este método retorna por padrão uma chave de 16 ***bytes*** (128 ***bits***), podendo ser modificado para 24 e 34 ***bytes*** por meio do método ***init()***.

```
chaveAES = keygenerator.generateKey();
cifraAES = Cipher.getInstance("AES/CBC/PKCS5Padding");
vetorI = new IvParameterSpec(IV.getBytes("UTF-8"));
```

Quadro 4 - Alteração no código de Higor Medeiros para gerar a chave e a cifra
Fonte: Adaptado de Medeiros (2014)

Uma das formas de utilizar uma chave diferente a cada processo de criptografia é implementar os métodos padrões do pacote javax.crypto do Java. Logo após a criação da chave e da cifra foi gerado o vetor de inicialização por meio do método ***IvParameterSpec()*** a partir da ***string*** IV.

Na criação da cifra não foi especificado o provedor do qual se deseja utilizar a implementação, então a própria classe retorna a implementação do primeiro provedor encontrado em uma lista de possíveis fornecedores que implementam o algoritmo AES.

O vetor de inicialização introduz uma maior aleatoriedade ao código, aumentando assim a segurança do algoritmo. É necessário que o vetor seja do mesmo tamanho do bloco utilizado e que seja igual tanto na criptografia como na decriptografia.

Na versão original do código, o autor cria o vetor de inicialização no momento da inicialização da cifra para encriptar (***ENCRYPT_MODE***) ou decriptar

(***DECRYPT_MODE***) a mensagem com a chave secreta, conforme demostra o quadro abaixo. O método ***doFinal()*** é o responsável por encriptar/decriptar a mensagem.

```
cifraAES.init(Cipher.ENCRYPT_MODE, chaveAES, new IvParameterSpec (IV.getBytes ("UTF-8")));
encripta.doFinal(textopuro.getBytes("UTF-8"));
```

```
cifraAES.init(Cipher.DECRYPT_MODE, chaveAES, new IvParameterSpec (IV.getBytes ("UTF-8")));
String(decripta.doFinal(textoencriptado),"UTF-8");
```

Quadro 5 - Código para encriptação e decriptação no algoritmo AES
Fonte: Medeiros (2014)

Uma das modificações realizadas na forma de encriptar e decriptar, foi o fato de que no código modificado já havia sido previamente criado o vetor de inicialização juntamente com a chave e a cifra, no momento da inicialização ele é passado como parâmetro.

```
cifraAES.init(Cipher.ENCRYPT_MODE, chaveAES, vetorI);
outStream.write(cifraAES.update(buffer, 0, len));
outStream.write(cipher.doFinal());
```

```
cifraAES.init(Cipher.DECRYPT_MODE, chaveAES, vetorI);
outStream.write(cifraAES.update(buffer, 0, len));
outStream.write(cipher.doFinal());
```

Quadro 6 - Alteração no código de Higor Medeiros para encriptação e decriptação
Fonte: Adaptado de Medeiros (2014)

A criação da variável ***outStream*** é explicada com detalhes no capítulo 2.7. O método ***update()*** foi implementado para criptografar/descriptograr os arquivos de imagem, ele se encontra dentro de um laço de repetição para que possa “alcançar” todos os blocos de ***bytes*** nos quais a imagem foi dividida e o método ***doFinal()*** encerra o processo de criptografia/decriptografia.

O desempenho do algoritmo AES em tempo e tamanho de arquivos gerados são apresentados no capítulo 4.2.

2.6.2 Blowfish

Para criptografar uma mensagem por meio do algoritmo Blowfish são utilizadas as seguintes funções: Feistel iterativa de 16 rodadas realizando XOR em ***bits***, consulta à tabela e adição; quatro ***S-Boxes*** (caixas de substituição para embaralhar os dados) de tamanho 8x32; geração de 18 subchaves por iterações a partir da chave principal.

A chave deste algoritmo pode ter o tamanho máximo de 448 ***bits*** e o conjunto de subchaves pode ter no máximo 4168 **bytes** (33344 ***bits***).

> A cifragem dos dados ocorre através de uma rede de Feistel com 16 iterações. Essa etapa utiliza um método de cifragem fraco através de várias iterações, de modo a tornar-se um processo complexo. Cada iteração consiste de uma permutação dependente somente da chave e de uma substituição dependente da chave e dos dados envolvidos. Todas as operações realizadas são XORs (Ou Exclusivos) e adições sobre palavras de 32 ***bits***. As funções utilizadas nas iterações, XORS e adições, são funções simples e eficientes quando executadas pelos microprocessadores (BUGATTI, 2005, p. 44).

O método de criptografia por meio do Blowfish utilizado neste trabalho foi implementado por Dhanoop Bhaskar, em seu artigo "***Java – Encryption and Decryption of an Image Using Blowfish Algorithm***".

Apesar deste algoritmo trabalhar com arquivos – que é o escopo deste trabalho – algumas modificações tiveram de ser feitas, adicionando algumas funcionalidades que não foram implementadas no código original.

Para gerar a chave, Dhanoop Bhaskar implementou o algoritmo Blowfish com métodos similares a versão final implementada do AES. No quadro abaixo é demonstrado o código para geração da chave.

```
keyGenerator = KeyGenerator.getInstance("Blowfish");
secretKey = keyGenerator.generateKey();
cipher = Cipher.getInstance("Blowfish");
```

Quadro 7 - Código para geração da chave e da cifra algoritmo *Blowfish*
Fonte: Bhaskar (2013)

O autor realizou a instanciação do algoritmo Blowfish, gerou a chave por meio do método ***generateKey()*** e criou a cifra pra criptografia/decriptografia instanciando o algoritmo desejado.

As modificações realizadas no código acima podem ser vistas no quadro abaixo. Na primeira linha foi inserido o modo de cifra CBC e o PKCS5Padding que será

o responsável por completar os ***bytes*** do último bloco caso o tamanho dos dados a serem criptografados não sejam múltiplos de 8 (que é o padrão do algoritmo).

Esta funcionalidade exige que seja gerado um vetor de inicialização a ser enviado ao método padrão, juntamente com a mensagem e a chave.

```
cipher = Cipher.getInstance("Blowfish/CBC/PKCS5Padding");
vetorI = new IvParameterSpec(IV.getBytes("UTF-8"));
```

Quadro 8 - Alteração no código de Dhanoop Bhaskar para geração da cifra
Fonte: Adaptado de Bhaskar (2013)

O modo de cifra de CBC realiza um XOR em cada bloco de texto puro com o vetor de inicialização definido, sendo assim, o mesmo bloco de texto simples não resultará no mesmo bloco de texto cifrado.

Este modo somente é vantajoso quando há grandes quantidades de dados a serem cifrados (exemplo: arquivos). Para quantidades pequenas, como na criptografia de textos curtos, outros modos de cifras são indicados.

```
cipher.init(Cipher.ENCRYPT_MODE, secretKey);
outStream.write(cipher.update(buffer, 0, len));
outStream.write(cipher.doFinal());
```

```
cipher.init(Cipher.DECRYPT_MODE, secretKey);
outStream.write(cipher.update(buffer, 0, len));
outStream.write(cipher.doFinal());
```

Quadro 9 - Código para encriptação e decriptação algoritmo *Blowfish*
Fonte: Bhaskar (2013)

Para criptografar e decriptografar Dhanoop Bhaskar cria dois métodos e em cada um deles utiliza o método ***init()***, inicializando o objeto para sua determinada função (***ENCRYPT*** ou ***DECRYPT***) com a chave.

Utiliza-se a variável ***outStream*** (ver capítulo 2.7) para criptografar/decriptografar cada bloco do arquivo de imagem pelo método ***update()*** e finaliza o processo por meio do método ***doFinal()***.

A única alteração nos métodos de criptografia e decriptografia foi realizada por causa da imposição (pelo modo de cifra CBC) da utilização do vetor de inicialização.

```
cipher.init(Cipher.ENCRYPT_MODE, secretKey, vetorI);
cipher.init(Cipher.DECRYPT_MODE, secretKey, vetorI);
```

Quadro 10 - Alteração do código de Bhaskar para encriptação e decriptação
Fonte: Adaptado de Bhaskar (2013)

Os testes realizados com o algoritmo Blowfish são apresentados no capítulo 4.3.

2.6.3 Rivest Ciphers 2

O algoritmo RC2 é uma cifra de blocos utilizado no protocolo S/MIME – voltado para criptografia de ***e-mail*** corporativo. Este algoritmo pode substituir o algoritmo DES com maior segurança, pois possui tamanho de chave variável e segundo a literatura chega a ser duas vezes mais veloz que o algoritmo DES.

Assim, como os outros algoritmos de criptografia, realiza funções matemáticas com a chave, a mensagem e o vetor de inicialização até chegar a mensagem criptografada/decriptografada.

> RC2 - este algoritmo de bloco foi desenvolvido originalmente por Ronald Rivest, e mantido em segredo pela RSA Data Security. Foi revelado por uma mensagem anônima na ***Usenet*** em 1996, e parece ser relativamente poderoso (embora algumas chaves sejam vulneráveis). O RC2 é vendido com uma implementação que permite a utilização de chaves de 1 a 2048 ***bits*** (SILVA, 2004, p. 08).

A implementação do algoritmo RC2 neste trabalho segue o mesmo padrão utilizado nos algoritmos anteriores. A forma de gerar a chave, o vetor de inicialização, a cifra e a forma de criptografar/decriptografar seguem a mesma implementação, a única diferença são os parâmetros que são enviados aos métodos, os quais são referentes ao algoritmo em questão.

No quadro abaixo é demonstrado como são passados os parâmetros às classes da API de criptografia.

```
keyGenerator = KeyGenerator.getInstance("RC2");
cipher = Cipher.getInstance("RC2/CBC/PKCS5Padding");
```

Quadro 11 - Geração da chave e da cifra com o algoritmo RC2
Fonte: Autoria Própria

Os testes realizados com essa implementação do algoritmo RC2 são detalhados no capítulo 4.4.

2.6.4 Rivest Ciphers 5

É a quinta cifra projetada pelo professor Ronald L. Rivest, publicada no ano de 1994. “Esse algoritmo usa rotação dependente de dados como sua operação não linear e é parametrizado de modo que o usuário possa variar o tamanho do bloco, o número de estágios e o comprimento da chave (BUGATTI, 2005, p. 23).”

Para seguir os padrões deste trabalho, o tamanho da chave no algoritmo RC5 é alterado por meio do método ***init()***. A chave pode possuir o tamanho de 128, 192 ou 256 ***bits***. Já para o tamanho do bloco e para o número de estágios, são utilizados o padrão implementado pela API de criptografia.

Para realizar suas funções o “RC5 baseia-se na operação de rotação (i.e., deslocamento circular) de um número variável de posições, e esse número depende de quase todos os ***bits*** resultantes da iteração anterior e do valor da subchave em cada iteração” (MORAES, 2004, p. 118).

A forma de gerar a chave, o vetor de inicialização, a cifra, o método para criptografia e o método de decriptografia com o RC5 são exatamente iguais aos demais algoritmos, só diferem-se os parâmetros repassados para o método ***getInstance()***, como é demonstrado no quadro abaixo:

```
keyGenerator = KeyGenerator.getInstance("RC5");
cipher = Cipher.getInstance("RC5/CBC/PKCS5Padding");
```

Quadro 12 - Geração da chave e da cifra com o algoritmo RC5
Fonte: Autoria Própria

Após realizada a chamada ao método de geração da chave, o programa retornou o seguinte erro:

```
run:
Exception in thread "main" java.security.NoSuchAlgorithmException: RC5 KeyGenerator not available
        at javax.crypto.KeyGenerator.<init>(KeyGenerator.java:169)
        at javax.crypto.KeyGenerator.getInstance(KeyGenerator.java:223)
        at Simetrico.EncriptaDecriptaRC5.<init>(EncriptaDecriptaRC5.java:28)
        at Simetrico.Teste.main(Teste.java:20)
Java Result: 1
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Figura 4 - Erro ocorrido na geração da chave com o algoritmo RC5
Fonte: Autoria Própria

Isto ocorreu porque a classe ***KeyGenerator*** não implementa o algoritmo. Como no escopo do trabalho foi definido que seria testado o RC5, foi necessária a importação do fornecedor ***Bouncy Castle***. Este fornece um pacote que contém a implementação em Java de algoritmos de criptografia por meio de interfaces de programação de aplicativos (API's).

Ele pode ser utilizado em qualquer ambiente de desenvolvimento e é livre de licenças. O quadro abaixo demonstra a implementação do código para a utilização desta API.

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;
Security.addProvider(new BouncyCastleProvider());
```

Quadro 13 - Importação e utilização do provedor *Bouncy Castle*
Fonte: Fonseca (2010)

É importante frisar que com a importação deste pacote é possível utilizar as classes para geração da chave e da cifra de acordo com a especificação da JCE normalmente, como foi feito nos demais algoritmos. Os testes realizados com o RC5 são apresentados no capítulo 4.5.

2.6.5 Rivest Ciphers 4

A quarta cifra projetada por Ron Rivest. O algoritmo RC4 é o único simétrico que não utiliza cifra em blocos e sim cifras de fluxo – a cada ***byte*** de saída do algoritmo são realizadas de oito a dezesseis operações de máquina. Ele é implementado em redes ***Wireless*** no padrão WEP com chaves que variam de 40 a 128 ***bits***.

Segundo a literatura este algoritmo pode chegar a ser dez vezes mais rápido que o algoritmo DES. Além disso ele proporciona a escolha do tamanho da chave, proporcionando maior segurança sobre a chave de tamanho fixo do DES.

> Nenhuma das técnicas de ataque publicadas em ***Knudsen*** (1998), ***Mister*** (1998), ***Fluhrer*** (2000) e ***Mantim*** (2001) são práticas contra esse algoritmo utilizando um tamanho de chave razoável, como 128 ***bits***. Apesar de terem sido detectados problemas relacionados à sua utilização no protocolo WEP, que oferece confidencialidade em redes locais sem fio 802.11, quanto à distribuição de chaves, esse problema específico não parece ser relevante a outras aplicações que utilizam o algoritmo (MOREIRA, 2010, p. 74).

Como todos os algoritmos de criptografia o RC4 realiza operações matemáticas com a chave e o texto claro da mensagem (a imagem original, no caso deste trabalho). Ele consiste de duas fases: a primeira transforma a chave numa permutação do conjunto de ***bytes***, a segunda usa essa permutação para gerar uma sequência pseudoaleatória de ***bytes***.

Para criptografar, basicamente é realizado um XOR entre os ***bytes*** da chave e os ***bytes*** originais. Para decriptografar é feito um XOR entre os ***bytes*** da mensagem criptografada e os ***bytes*** da chave, resultando na mensagem clara. Ele gera sequências pseudoaleatórias fortes, com simplicidade e rapidez a partir de suas permutações e somas de valores inteiros.

A implementação do algoritmo RC4 neste trabalho segue os mesmos padrões dos algoritmos anteriormente explicados. A forma de gerar a chave e a forma de criptografar/descriptografar é a mesma.

A diferença é que por padrão neste algoritmo não se utiliza o modo de cifra CBC, nem o Padding – pois ele não trabalha com blocos e sim com fluxo – então passa-se como parâmetro para a geração da cifra somente o nome do algoritmo como é feito na geração da chave.

Além disso, não é necessário gerar o vetor de inicialização. No quadro abaixo é mostrado como são geradas a cifra e a chave.

```
keyGenerator = KeyGenerator.getInstance("RC4");
cipher = Cipher.getInstance("RC4");
```

Quadro 14 - Geração da chave e da cifra com o algoritmo RC4
Fonte: Autoria Própria

No capítulo 4.6 são detalhados os testes realizados com a implementação deste algoritmo.

2.6.6 Data Encryption Standard e Triple Data Encryption Standard

O Data Encryption Standard é o um dos algoritmos mais simples de criptografia. Utiliza uma chave fixa de tamanho 56 ***bits*** (o tamanho real dela é de 64 ***bits*** – existem 8 ***bits*** de paridade) e um bloco de 64 ***bits***. Abaixo elenca-se o funcionamento da função de criptografia deste algoritmo:

> É feita uma permutação inicial no bloco de 64 ***bits*** e depois o bloco é dividido em duas metades. Após a divisão, o seguinte procedimento é repetido dezesseis vezes: 1. Uma metade do bloco serve de entrada para a função de embaralhamento (função de Feistel) juntamente com uma sub-chave de 48 ***bits*** derivada da chave principal de 56 ***bits***. Esta mesma metade é enviada para a próxima rodada e será entrada para o XOR juntamente com a saída da função de Feistel. Existe um total de 16 sub-chaves, uma para cada rodada; 2. A função de Feistel expande os 32 ***bits*** de entrada para 48 ***bits*** e faz um XOR com a sub-chave; 3. Os 48 ***bits*** de saída do XOR são separados em oito conjuntos de seis ***bits***; 4. É feita então uma transformação não linear que transforma os seis ***bits*** de cada conjunto em quatro ***bits*** de saída para cada conjunto; 5. Finalmente os 32 ***bits*** de saída do passo anterior sofrem uma permutação e este é o resultado de saída da função; 6. É feito um XOR entre a saída da função de Feistel e a outra metade do bloco de entrada do passo 1. Após as dezesseis rodadas é feita uma permutação final no bloco de 64 ***bits*** gerando o texto cifrado (BRAZIL, 2007, p. 20 e 21).

Para gerar o texto decifrado o algoritmo utiliza as mesmas funções, mas possuindo como entrada as subchaves em ordem inversa.

O algoritmo DES foi implementado neste trabalho, seguindo os mesmos padrões dos demais para a criação da chave, do vetor de inicialização, da cifra, do método de criptografia e da decriptografia, utilizando as classes da API de criptografia do Java.

As únicas modificações foram os parâmetros repassados às classes, que seguem o padrão do algoritmo em questão. No quadro abaixo são demonstradas as chamadas aos métodos de geração da chave e da cifra.

```
keygenerator = KeyGenerator.getInstance("DES");
cifraDES = Cipher.getInstance("DES/CBC/PKCS5Padding");
```

Quadro 15 - Geração da chave e da cifra com o algoritmo DES

Fonte: Autoria Própria

O algoritmo 3DES é uma variação do algoritmo DES. Ele utiliza três conjuntos distintos da chave de 56 ***bits***, totalizando 168 ***bits*** (mais 24 ***bits*** de paridade, totalizando uma chave de 192 ***bits***), e um bloco de tamanho 64 ***bits***.

"O DES triplo utiliza o sistema EDE para cifragem, ou seja, o texto plano é encriptado primeiro com a primeira chave, decriptado com a segunda e encriptado novamente com a terceira (MORAES, 2004, p. 62)."

Para a utilização do algoritmo 3DES as únicas alterações realizadas foram os parâmetros repassados às classes da API de criptografia do Java. No quadro abaixo é demonstrada a chamada aos métodos para a criação da chave e da cifra, nota-se que é passado como parâmetro a ***string*** "DESede" que corresponde à implementação do algoritmo 3DES na API (ver capítulo 2.5).

```
keygenerator = KeyGenerator.getInstance("DESede");
cifraTDES = Cipher.getInstance("DESede/CBC/PKCS5Padding");
```

Quadro 16 - Geração da chave e da cifra com o algoritmo 3DES
Fonte: Autoria Própria

Os testes realizados com a implementação dos algoritmos DES e 3DES são apresentados nos capítulos 4.7 e 4.8 respectivamente.

2.6.7 Ronald Rivest, Adi Shamir e Leonard Adleman

RSA tem sua sigla derivada dos nomes de seus criadores (Rivest, Shamir e Adleman), lançado no ano de 1978, é um algoritmo de chave assimétrica, ou seja, cada participante possui dois pares de chaves: uma pública e outra privada. A chave pública é utilizada para cifrar o texto a ser enviado e a chave privada é a responsável por decifrar o texto recebido encriptado.

> "O RSA funciona da seguinte forma: o emissor A descobre dois números primos grandes (por exemplo, 600 algarismos) *p* e *q* e calcula seu produto, logo *n* = *pq*. O receptor B deve de alguma forma também saber o valor de *n*. O emissor A também descobre dois inteiros *e* e *d*, sendo que o receptor B deverá saber o valor de *d*. Assim, esse é um algoritmo de criptografia assimétrica com uma chave pública PU = *{e,n}* e uma chave privada PR = *{d,n}* (MOREIRA, 2010, p. 68 e 69)."

Este algoritmo assim como os assimétricos utiliza uma cifra de blocos. Não há registros de ataques de "força bruta" no algoritmo RSA bem sucedidos em um curto espaço de tempo. Quanto maior a chave utilizada, maior a segurança deste algoritmo.

> "Quanto ao RSA e a outros algoritmos de chaves públicas, sua segurança tem como base a dificuldade envolvendo a fatoração de números primos grandes. Ao passo que é fácil multiplicar dois números primos grandes, fatorar o produto desses dois números é muito mais difícil. As chaves pública e privada do RSA são funções de pares de números primos muito grandes, com centenas de dígitos. Uma característica do RSA e de outros algoritmos de chave pública é que eles podem ser utilizados para a cifragem de dados e também para a autenticação por meio de assinaturas digitais (NAKAMURA E GEUS, 2007, p. 296)."

O código do algoritmo RSA utilizado neste trabalho foi implementado por Higor Medeiros em seu artigo "Criptografia assimétrica: criptografando e descriptografando dados em Java". Neste artigo o autor salva as chaves geradas em arquivos com extensão "***key***", e faz a criptografia e decriptografia de uma ***string*** de texto puro.

O quadro abaixo demonstra o código utilizado pelo autor para geração do par de chaves. As variáveis ***PATH_CHAVE_PRIVADA*** e ***PATH_CHAVE_PUBLICA*** são os caminhos onde as chaves serão salvas. Após gerar as chaves o programa salva-as nos arquivos correspondentes.

```
public static final String ALGORITHM = "RSA";
public static final String PATH_CHAVE_PRIVADA = "C:/keys/private.key";
public static final String PATH_CHAVE_PUBLICA = "C:/keys/public.key";
final KeyPairGenerator keyGen = KeyPairGenerator.getInstance(ALGORITHM);
keyGen.initialize(1024);
final KeyPair key = keyGen.generateKeyPair();
```

Quadro 17 - Código de geração do par de chaves com RSA
Fonte: Medeiros (2014)

Para a realização deste trabalho algumas alterações foram feitas no código de Higor Medeiros, para que este algoritmo tivesse o mesmo padrão de implementação dos algoritmos simétricos, tendo em vista suas particularidades de assimétrico.

Nota-se que para o trabalho, assim como no código do autor, foi gerado o par de chaves por meio do método ***generateKeyPair()***, mas ao invés de salvá-las em um arquivo, cada uma foi recuperada em uma ***string*** correspondente através dos métodos ***getPublic()*** e ***getPrivate()***.

É importante frisar que as chaves não foram salvas em arquivos porque o trabalho não aborda a transmissão da chave, são feitos somente os testes de desempenho dos algoritmos. Ao contrário do código do autor, a cifra também é gerada junto à chave.

As alterações realizadas no código para geração do par de chaves podem ser vistas no quadro abaixo.

```
key = key.getInstance("RSA");
par = key.generateKeyPair();
chavePublica = par.getPublic();
chavePrivada = par.getPrivate();
cifra = Cipher.getInstance("RSA");
```

Quadro 18 - Alteração no código de Medeiros para geração das chaves e da cifra
Fonte: Adaptado de Medeiros (2014)

Para encriptar e decriptar a mensagem, Higor Medeiros cria a cifra nas suas determinadas funções e a utiliza para encriptar (***ENCRYPT_MODE***) ou decriptar (***DECRYPT_MODE***) a mensagem que é passada como parâmetro para a classe.

```
final Cipher cipher = Cipher.getInstance(ALGORITHM);
cipher.init(Cipher.ENCRYPT_MODE, chave);
cipherText = cipher.doFinal(texto.getBytes());
```

```
final Cipher cipher = Cipher.getInstance(ALGORITHM);
cipher.init(Cipher.DECRYPT_MODE, chave);
dectyptedText = cipher.doFinal(texto);
```

Quadro 19 - Código para encriptação e decriptação com o algoritmo RSA
Fonte: Medeiros (2014)

A chave correspondente (privada – encriptar; pública – decriptar) também é passada como parâmetro. Ao final as informações correspondentes (texto codificado e texto decodificado) são geradas a partir do método ***doFinal()***. O código implementado para criptografia/decriptografia pode ser visto no quadro acima.

Foram realizadas alterações no código, tendo em vista que o autor codifica/decodifica texto puro, e este trabalho testa o desempenho dos algoritmos na criptografia de imagens.

Como foi citado anteriormente, ao contrário do autor, foi implementado neste trabalho a criação da cifra junto ao par de chaves. Outra diferença é a utilização do

método ***uptade()***, este método se encontra em um laço de repetição ***while()*** para que seja possível criptografar todos os ***bits*** constantes na imagem.

O processo é finalizado por meio do método ***doFinal()***. Já a variável ***outStream*** é a responsável pela escrita no arquivo correspondente (ver capítulo 2.7). As alterações no código de Higor Medeiros são demonstradas no quadro abaixo.

```
cifra.init(Cipher.ENCRYPT_MODE, chavePublica);
outStream.write(cifra.update(buffer, 0, len));
outStream.write(cifra.doFinal());
```

```
cifra.init(Cipher.DECRYPT_MODE, chavePrivada);
outStream.write(cifra.update(buffer, 0, len));
outStream.write(cifra.doFinal());
```

Quadro 20 - Alteração no código de Medeiros para criptografia e decriptografia
Fonte: Adaptado de Medeiros (2014)

Os testes realizados na implementação do algoritmo de criptografia assimétrico RSA são demonstrados no capítulo 4.9.

2.7 FUNÇÃO DE HASH

Uma função de resumo (***hash***) criptografa dados com tamanho fixo, ou seja, independentemente do tamanho da informação, a saída terá sempre o mesmo tamanho.

Esta encontra aplicações na verificação da integridade de um arquivo armazenado, verificação da integridade de um arquivo obtido na ***Internet***, geração de assinaturas digitais, etc.

> Para verificar a integridade de um arquivo, por exemplo, você pode calcular o ***hash*** dele e, quando julgar necessário, gerar novamente este valor. Se os dois ***hashes*** forem iguais então você pode concluir que o arquivo não foi alterado. Caso contrário, este pode ser um forte indício de que o arquivo esteja corrompido ou que foi modificado. (CERT.br, 2012, p. 69).

O ***hash*** é implementado de tal forma que uma mesma informação sempre produzirá o mesmo resultado, mas diferentemente da criptografia simétrica e

assimétrica, não é possível fazer o processo inverso para obtenção da informação original.

Se houver alguma modificação na informação original, o ***hash*** retornará um valor distinto, por isso é possível comparar a integridade do arquivo. Há a possibilidade de duas informações distintas gerarem o mesmo ***hash***, mais a probabilidade disto ocorrer é bastante baixa.

"O valor de ***hash*** é anexado à mensagem no momento em que ela é considerada como correta, ou seja, quando ela ainda não foi alterada por terceiros. O receptor autentica essa mensagem recalculando o valor de ***hash*** (MOREIRA, 2010, p. 71)".

Algumas das funções de ***hash*** mais conhecidas são: SHA-1, SHA-256, MD4 e MD5. O SHA-1 é baseado no MD4, mas provê uma maior segurança em comparação ao seu antecessor contra ataques de força bruta por utilizar um valor de 160 ***bits***.

Apesar disto, após a introdução do AES com chaves de tamanho 128, 192 e 256 ***bits***, esta função começou a apresentar falhas e colisões, sendo mais viável a utilização do SHA-256.

> O ***Secure Hash Algorithm*** (SHA-1), uma função de espalhamento unidirecional inventada pela NSA, gera um valor ***hash*** de 160 ***bits***, a partir de um tamanho arbitrário de mensagem. O funcionamento interno do SHA-1 é muito parecido com o observado no MD4, indicando que os estudiosos da NSA basearam-se no MD4 e fizeram melhorias em sua segurança. De fato, a fraqueza existente em parte do MD5, descoberta após o SHA-1 ter sido proposto, não ocorre no SHA-1. Em 2005, falhas de segurança foram identificados no SHA-1, ou seja, que uma fraqueza matemática pode existir, o que indica que o uso de uma função ***hash*** mais forte é recomendável, o que motiva o uso preferencial de SHA-2 (OLIVEIRA, 2012, p. 07).

O SHA-256 é a evolução do SHA-1. Também conhecido como SHA-2, foi criado porque seu antecessor não apresentava mais a segurança necessária. Esta função tem sido amplamente utilizada em certificações digitais.

> O ***Secure Hash Algorithm*** (SHA-2) por outro lado significativamente difere da função ***hash*** SHA-1, desenhado pelo NSA é uma família de duas funções ***hash*** similares, com diferentes tamanhos de bloco, conhecido como SHA-256 e SHA-512. Eles diferem no tamanho, o SHA-256 utiliza 256 bits e o SHA-512 utiliza 512 bits. Há também versões truncadas de cada padrão, conhecidos como SHA-224 e SHA-384. O ICP-Brasil em suas mudanças anunciadas adotadas para o novo padrão criptográfico do sistema de certificação digital, esta implantando em 2012, o uso do SHA-512 em substituição ao seu antecessor, o SHA-1 (OLIVEIRA, 2012, p. 7).

O MD4 foi criado para ser computacionalmente inviável quebrar seu resultado por força bruta, mas não alcançou seu objetivo, apresentando falhas e colisões em uma de suas funções.

Já o MD5 foi criado para suprir as necessidades de seu antecessor, mas também apresentou falhas e não conseguiu alcançar o nível de segurança almejado com sua criação.

> As funções de ***hash*** MD (***Message Digest***) foram propostas por Rivest e são utilizadas por diversas aplicações, entre elas pelo SNMP (***Secure Network Management Protocol***), pelo PGP e por muitos dos esquemas de assinaturas digitais utilizados. A função proposta inicialmente foi a MD2, havendo então uma evolução contínua até a MD5, a versão mais recente. O MD4 é o mais rápido e o MD5 é uma forma mais segura do MD4, porém mais lenta. Todos os três algoritmos recebem como entrada mensagens de tamanho arbitrário e produzem valores ***digest*** de 128 bits. Apesar dos MDs possuírem estruturas similares, o projeto do MD2 é muito diferente daquele do MD4 e do MD5. MD2 foi otimizado para máquinas de 8 bits, ao passo que MD4 e MD5 foram feitos para máquinas de 32 bits (MIRANDA, 2002, p. 36).

Funções de ***hash*** devem ser combinadas a algum algoritmo de criptografia para poder ser utilizada, geralmente um algoritmo assimétrico, como é o caso da certificação digital, para certificar ao receptor que a mensagem não foi modificada durante a comunicação.

Neste trabalho não serão implementadas nenhuma das funções ***hash*** porque o escopo é testar o desempenho dos algoritmos na criptografia e na decriptografia de imagens, sendo necessária a recuperação da informação no fim da comunicação – a função ***hash*** não permite a recuperação da informação clara a partir de seu valor final.

2.8 CRIPTOGRAFIA DE IMAGENS

O processamento digital de imagem do mais genérico ao mais sofisticado pode ser descrito pela imagem abaixo.

A aquisição da imagem é feita por dispositivos que as captam; um determinado computador é responsável pelo processamento inicial desta imagem – o termo computador pode ser trocado por processador; seu armazenamento pode ser feito em discos ópticos, fitas e discos magnéticos, ***pendrives***, banco de dados, etc.; a saída é o

ponto que receberá essas informações logo após seu processamento, pode ser o monitor do próprio processador ou então outro computador, impressoras, etc.

É muito comum enviar as imagens do ponto de processamento ao ponto de saída via rede interna ou externa.

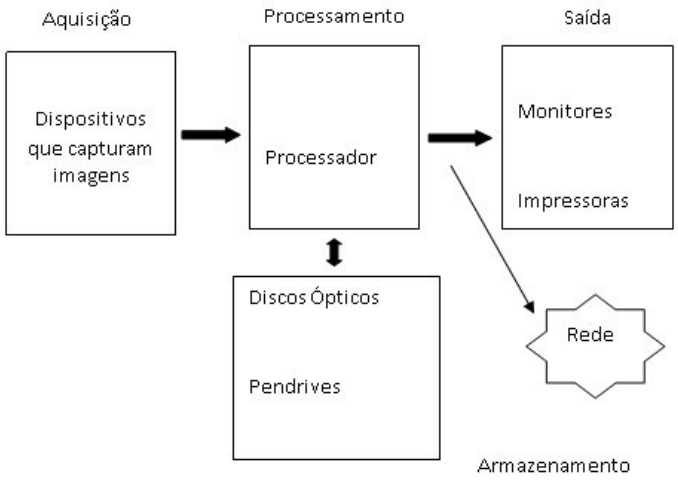

Figura 5 - Esquema de processamento de imagens
Fonte: Adaptado de Filho e Neto (1999)

A imagem a ser transportada e/ou arquivada pode ser confidencial. Assim como no caso das mensagens, são necessárias implementações de criptografia para assegurar que as imagens cheguem ao seu destino final (rede ou armazenamento) íntegras e com nenhuma perda, caso sejam interceptadas no canal durante a comunicação ou algum intruso acesse o local onde elas estão armazenadas.

Além disso deve ser possível recuperar as imagens criptografadas sem que o processo tenha as danificado. Assim como na criptografia de mensagens, são utilizados algoritmos para “embaralhar” a imagem.

> As técnicas de criptografia em imagens encontram aplicação em ambientes em que imagens confidenciais, por exemplo, imagens médicas, contratos, escrituras, mapas entre outros, precisam ser armazenadas ou transmitidas através de um canal de comunicação inseguro, a ***internet*** (SILVA ET AL, 2013, p. 02).

Dados de texto puro são criptografados nos algoritmos por meio de funções matemáticas que invertem, somam, multiplicam, etc. seus ***bits*** correspondentes. Nas

imagens não é diferente, elas são submetidas a várias funções matemáticas (as mesmas utilizadas em texto) até se tornarem imagens criptografadas.

Para realizar um processamento de imagens é importante utilizar um algoritmo que seja eficiente, pois serão criptografados vários blocos de ***bits*** simultaneamente, e esses blocos de informação criptografada devem ser agrupados de tal forma que no momento em que eles forem decriptografados estejam na ordem correta, para não corromper e inutilizar a imagem.

"Uma imagem é composta por uma quantidade grande de ***pixels***, dessa forma é imprescindível a escolha de um algoritmo criptográfico adequado que garanta a eficiência da velocidade × segurança." (SILVA ET AL, 2013, p. 06).

A criptografia de imagens utilizada neste trabalho é demonstrada nos quadros abaixo e foi implementada por Dhanoop Bhaskar, em seu artigo "***Java – Encryption and Decryption of an Image Using Blowfish Algorithm***".

O autor implementou a criptografia de imagens utilizando o algoritmo Blowfish, mas a mesma implementação de leitura, acesso aos ***bits*** e gravação pode ser utilizada para todos os algoritmos testados neste trabalho. A variável ***derectoryPath*** contém o caminho principal onde estarão salvas as pastas das imagens (originais, criptografadas e decriptografadas).

A chamada aos métodos de criptografia e decriptografia recebem como parâmetro essas variáveis de acordo com sua utilização.

```
String directoryPath;
String fileToEncrypt;
String encryptedFile;
String decryptedFile;
```

```
encrypt(directoryPath + imgOriginal, directoryPath + imgEncriptada);
decrypt(directoryPath + imgEncriptada, directoryPath + imgDecriptada);
```

Quadro 21 - Métodos para criptografar/decriptografar imagens
Fonte: Bhaskar (2013)

A variável ***fileToEncrypt*** contém o caminho específico da imagem a ser criptografada; a ***encryptedFile*** contém o caminho da pasta onde serão salvas (e como serão nomeadas) as imagens criptografadas, esta mesma variável indicará ao método de decriptografia onde foram salvas as imagens criptografadas; a ***decryptedFile***

contém o caminho especifico onde serão salvas (e como serão nomeadas) as imagens descriptografadas.

Os métodos descritos anteriormente são representados no quadro a seguir. O método ***encrypt*** é o responsável pela encriptação e o ***decrypt*** pela decriptação.

O parâmetro ***srcPath*** utilizado no método ***encrypt*** é o caminho onde se encontra a imagem original, seu nome e sua extensão; a utilização deste mesmo parâmetro no método ***decrypt*** indica o caminho onde se encontra a imagem encriptada, seu nome e sua extensão.

O parâmetro ***destPath*** utilizado no método ***encrypt*** é o caminho onde será gravada a criptografia, o nome e a extensão que será atribuída ao arquivo; a utilização deste mesmo parâmetro no método ***decrypt*** indica o caminho onde será gravada a decriptografia, o nome e a extensão que será atribuída ao arquivo.

```
encrypt(String srcPath, String destPath){}
decrypt(String srcPath, String destPath){}
```

Quadro 22 - Métodos de criptografia/decriptografia e seus parâmetros
Fonte: Bhaskar (2013)

Para criptografar uma imagem é necessário obter cada um de seus ***bits***, e então codificá-los, este processo é demonstrado no quadro abaixo. A classe ***File*** do Java foi utilizada para representação do caminho onde são encontrados os arquivos.

A variável ***rawFile*** recebe o caminho da imagem a ser criptografada/decriptografada; a ***encryptedFile*** é utilizada no método de encriptação e recebe o caminho onde será salva a imagem encriptada; a ***decryptedFile*** é utilizada no método de decriptação e recebe o caminho onde será salva a imagem decriptada; a ***inStream*** é responsável pela leitura dos ***bytes*** da imagem; a ***outStream*** é a responsável pela escrita dos ***bytes*** no arquivo de destino.

São lidos todos ***bits*** dos ***bytes*** e criptografados/decriptografados todos os blocos gerados no laço ***while()***. Após o fim da criptografia/decriptografia (método ***doFinal()***), são encerrados o ***inputStream*** e o ***outputStream*** finalizando o processo.

A variável ***cipher*** é a cifra gerada a partir do algoritmo especificado, processo detalhado no capítulo 2.6. As modificações realizadas no algoritmo implementado por

Dhanoop Bhaskar foram as alterações nos caminhos das imagens e alguns nomes de variáveis.

É importante frisar que as classes ***FileInputStream*** e ***FileOutputStream*** fazem a leitura e a escrita em arquivos com qualquer extensão, não somente com arquivos de imagens.

```
File rawFile = new File(srcPath);
File encryptedFile = new File(destPath);
File decryptedFile = new File(destPath);
inStream = new FileInputStream(rawFile);
outStream = new FileOutputStream(encryptedFile);
byte[] buffer = new byte[1024];
int len;
while ((len = inStream.read(buffer)) > 0) {
        outStream.write(cipher.update(buffer, 0, len));
        outStream.flush();
}
outStream.write(cipher.doFinal());
inStream.close();
outStream.close();
```

Quadro 23 - Criptografia/decriptografia de imagens
Fonte: Bhaskar (2013)

Arquivos de imagens de extensão JPG são usualmente utilizados na ***Internet*** por ocuparem menor tamanho em disco se comparados com outras extensões. A possibilidade de compressão realizada pelo JPG faz-com que a qualidade das imagens seja reduzida, mas sem perdas significativas. O JPG é capaz de criar blocos de ***pixels***, ou seja, menos informações a serem armazenadas e/ou transmitidas pela rede.

Os dados criptografados poderiam ser salvos em arquivos de qualquer extensão, mas tendo em vista as vantagens de se trabalhar com arquivos de imagens JPG, este será o padrão adotado neste trabalho.

Usualmente os dados a serem enviados em um canal são arquivos de texto, mas segundo os resultados apresentados pelos autores do artigo “Criptografia assimétrica de imagens utilizando algoritmo RSA”:

> A geração do arquivo de texto resultava em um arquivo maior do que a geração da imagem. Uma imagem cifrada utilizando o método de gerar o arquivo de texto a cifragem tinha um resultado com 9MB de tamanho em disco, enquanto no método de gerar a imagem a mesma imagem cifrada tinha 1,26MB (SILVA ET AL, 2013, p. 04).

Os autores ainda citam que o tempo decorrido para gravar um arquivo de imagem era três vezes mais rápido do que gerar um arquivo de texto com os dados criptografados. No artigo acima citado, os autores criptografaram cada banda de RGB da imagem e geraram as imagens a partir delas utilizando o algoritmo RSA.

Já neste trabalho foram criptografados os ***bits*** da imagem retornados à partir de uma função da linguagem Java, fazendo uma comparação de alguns dos algoritmos de criptografia.

O escopo do artigo referenciado é o mesmo deste trabalho – medir tempo e eficiência, então seus resultados foram considerados, contribuindo para bem embasar a utilização da extensão JPG para gerar os arquivos criptografados/decriptografados neste trabalho.

Três pontos são importantes para a escolha do melhor algoritmo de criptografia de imagem: tempo decorrido para criptografia/decriptografia, alteração nos ***bytes*** do arquivo criptografado e integridade do arquivo descriptografado.

Foram testados estes três fatores neste trabalho, em cada um dos algoritmos implementados (ver capítulo 2.6). Os resultados são detalhados no capítulo 4.

3 ESTUDO DE CASO – IMAGENS DE RADAR DE TRÂNSITO

Este trabalho tem como escopo a comparação de alguns algoritmos simétricos e assimétricos para criptografia de imagens, no caso específico de imagens captadas por radares de trânsito.

São equipamentos de fiscalização instalados nas vias de rolamento para controlar a velocidade e aumentar a segurança de motoristas e pedestres. Cada equipamento possui um conjunto de câmera, processador e meio de comunicação (rede cabeada ou não), que são os responsáveis pela captura e o envio das imagens dos veículos que transitam com velocidade superior à permitida.

Estes equipamentos podem ser divididos em três tipos: radar fixo, radar estático e lombada ou barreira eletrônica. Para o desenvolvimento deste trabalho serão utilizadas imagens capturadas por um determinado radar fixo.

> O radar fixo e a barreira eletrônica apresentam sistema de detecção de velocidade baseado na tecnologia de laços indutivos instalados no pavimento. Normalmente é utilizado um conjunto de 2 ou 3 laços indutivos por faixa de rolamento. O valor da velocidade é obtida pelo quociente da distância entre os laços pelo tempo gasto para percorrê-la. No caso de 2 laços, o valor de velocidade pode ser obtido através de uma única medida ou por meio de duas medidas de tempo. No primeiro caso, é acionado um cronômetro quando o primeiro laço é sensibilizado pelo veículo. O cronômetro é travado quando o veículo alcança o segundo laço. No segundo caso, a primeira medida de tempo é feita entre os instantes de chegada do veículo no primeiro e segundo laço, enquanto que a segunda medida de tempo é feita entre os instantes de saída do primeiro e segundo laço. No caso do uso de três laços, a velocidade é sempre obtida por meio de duas medidas de tempo: entre o primeiro e segundo laço e entre o segundo e terceiro laço (MING, 2006, p. 2).

As imagens podem ser capturadas durante o dia ou durante a noite, além disso há radares que capturam a placa da frente do veículo e outros que capturam a placa de traz. Para não ofuscar o motorista e para uma qualidade similar com a da imagem capturada durante o dia, o ***flash*** da câmera durante a noite é infravermelho. Os radares fixos são capazes de detectar a velocidade de dois carros simultaneamente.

O envio das informações obtidas pelo radar até o ponto que processará esta informação é feita pela rede (local ou sem fio). Este é um meio inseguro, pois qualquer pessoa pode interceptar essa comunicação e alterar os dados, então, faz-se necessária a utilização de alguma forma para proteção desses dados.

> Por possuírem instalações fixas, é possível efetuar a transmissão remota das imagens registradas pelo radar fixo e barreira eletrônica, utilizando-se de conexões de banda larga como ADSL ou rádio comunicação. A possibilidade de transmissão de imagens é muito importante em grandes metrópoles como São Paulo, pois evita a necessidade de se deslocar periodicamente até cada equipamento para coletar as imagens registradas. Com a comunicação, há também a grande vantagem de se poder configurar o equipamento à distância, a partir de uma Central de Controle, sem a necessidade de se deslocar fisicamente até cada equipamento (MING, 2006, p. 3).

Uma das formas de proteção dos dados é a criptografia. Se a imagem for interceptada e o atacante não conseguir a senha utilizada na criptografia, não poderá manipulá-la. A proposta deste trabalho é encontrar um algoritmo que melhor se apresente na criptografia de imagens capturadas por radares de trânsito, com a maior segurança possível.

A imagem de trânsito para ter validade deve possuir alguns requisitos como boa resolução, nitidez e bom enquadramento do veículo, pois se não for possível visualizar a placa ou determinar a marca e o modelo do veículo não há como processar a multa. Por isso é necessário que o algoritmo implementado na criptografia não corrompa a imagem e não torne inviável sua utilização.

> A captura e o processamento de imagens constitui uma etapa importante no processo de fiscalização. Não basta que o equipamento detecte a velocidade dos veículos com precisão ou que o sistema LAP leia corretamente as placas. Se não houver um sistema eficiente de captura de imagens, a fiscalização não será produtiva, pois boa parte das imagens registradas não poderá ser convertida em multas. A imagem deve ter boa resolução e nitidez e deve apresentar um bom enquadramento do veículo, de forma a poder identificar a sua placa, marca e modelo (MING, 2006, p.8).

Este trabalho verifica se as formas encontradas para a encriptação de texto puro, podem ou não ser adotadas na encriptação de imagens, se isto não acarretará em perda de ***pixels*** e distorção da imagem. No caso das infrações de trânsito o algoritmo escolhido deve ser o que apresenta o melhor desempenho, pois a imagem deve estar nítida para que a multa possa ser gerada.

As imagens utilizadas neste trabalho para os testes de criptografia possuem extensão JPG, todas possuem a mesma altura (480 ***pixels***) e a mesma largura (752 ***pixels***), mas, não possuem o mesmo tamanho em ***bytes***. As imagens variam de 9.900 a 68.304 ***bytes***.

Dois exemplos de imagens utilizadas são demonstradas abaixo, onde a primeira foi capturada durante o dia e possui um tamanho de 42.078 ***bytes*** e a segunda foi capturada durante a noite, possuindo um tamanho de 30.082 ***bytes***, são demonstradas também a altura (h) e a largura (l) das imagens em ***pixels***.

Figura 6 - Exemplo de imagem utilizada no trabalho capturada durante o dia
Fonte: Ming (2006)

Figura 7 - Exemplo de imagem utilizada no trabalho capturada durante a noite
Fonte: Ming (2006)

Este trabalho se preocupa principalmente com o desempenho na criação das imagens criptografadas e decriptografadas (perdas ou ganhos de ***pixels***), além disso, também há a preocupação com o desempenho de tempo decorrido e com o tamanho da chave a ser utilizada. Por isso são realizados testes em vários algoritmos, demonstrando as particularidades de cada um.

Ao final deste trabalho será possível concluir qual é a melhor forma de criptografar uma imagem, onde o algoritmo deve apresentar o melhor tempo decorrido com a melhor chave (a de tamanho 256 ***bits*** – tamanho máximo testado no trabalho – pois quanto maior a chave, maior a segurança) e um desempenho satisfatório quanto ao ganho ou perda de ***pixels*** durante o processo.

4 RESULTADOS OBTIDOS

Foram testados neste trabalho os algoritmos de criptografia simétricos AES, Blowfish, RC2, RC5, RC4, DES, 3DES e o assimétrico RSA. Três condições foram observadas: tempo decorrido para criptografia e decriptografia de acordo com a quantidade de imagens; tempo decorrido para criptografia e decriptografia de acordo com o tamanho da chave; alteração nos tamanhos dos arquivos de imagem criptografados e decriptografados.

Nos testes de tempo decorrido de acordo com o tamanho da chave, para os algoritmos simétricos AES, Blowfish, RC2, RC5 e RC4 foram padronizados os tamanhos de chave de 128, 192 e 256 ***bits*** para observar o desempenho de cada algoritmo dada a quantidade de imagens.

Para os algoritmos DES e 3DES só foi possível utilizar chaves de tamanho 64 e 192 ***bits*** respectivamente (estes dois algoritmos possuem tamanho de chave fixo).

O tamanho dos blocos utilizados nos algoritmos AES, Blowfish, RC2, DES e 3DES mantem o padrão da especificação de cada um. Para o algoritmo RC5 foi utilizado o padrão da API de criptografia (64 ***bits***) para todos os tamanhos de chave.

Os tamanhos de chave utilizados para os testes com o algoritmo RSA foram de 2048, 4096 e 16384 ***bits***. A primeira foi escolhida por ser mais usualmente utilizada, a segunda por ser a chave que substitui a primeira em alguns casos e a terceira foi a escolhida por ser o tamanho máximo suportado pelo algoritmo.

Para os testes de tempo decorrido por algoritmo foram utilizados cinco tipos de pacotes de imagens. O primeiro continha uma única imagem de tamanho 49.1 Kb, o segundo continha dez imagens com um tamanho total de 460 Kb, o terceiro cinquenta imagens totalizando 2.24 Mb de tamanho, o quarto cem imagens com um tamanho de 4.54 Mb e o quinto duzentas imagens com um total de 8.02 Mb de tamanho.

A seguir são mostrados os recursos utilizados e os resultados obtidos em cada um dos algoritmos testados. É importante frisar que os testes de tempo podem variar de acordo com da complexidade da chave gerada pela função ***generateKey()***, bem como, também varia de acordo com o processador utilizado.

4.1 RECURSOS UTILIZADOS

O computador utilizado para os testes foi um ***notebook Asus***, com as seguintes configurações:

- Processador ***Intel Core*** i5 – 3317U;
- ***CPU*** de 1.7GHz;
- Memória ***RAM*** de 8GB
- ***HD*** de 500GB
- Sistema operacional ***Windows*** 10.

Os códigos utilizados neste trabalho foram desenvolvidos na IDE ***NetBeans*** 8.0.2 portando a versão 1.8 do Java.

Para a utilização de chaves de tamanho 192 e 256 ***bits*** nos testes foi necessário fazer o ***dowload*** do pacote "***Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files***" específico para a versão oito do Java, diretamente do ***site*** da Oracle. Os arquivos do pacote foram descompactados na pasta "javahome/jre/lib/security/".

4.2 ALGORITMO AES

Como citado anteriormente foram analisadas algumas questões nos testes realizados com cada um dos algoritmos, são elas: tamanho da imagem criptografada, tamanho da imagem decriptografada, desempenho no tempo de acordo com o tamanho da chave e de acordo com a quantidade de fotos.

Na questão de integridade da informação (criptografada e decriptografada) o algoritmo AES apresentou resultados satisfatórios, pois não houve perda de dados durante o processo. Para uma amostragem foram escolhidas dez imagens aleatórias, as quais são apresentadas na tabela abaixo.

Nota-se que a diferença entre o tamanho da imagem original e da imagem decriptografada é zero, demonstrando que não houve perda de dados com o processo.

Nota-se também que em determinados casos houve um aumento maior na quantidade de **bytes** na imagem criptografada.

Tabela 2 - Tamanho da imagem em *bytes* durante o processo com AES

Tamanho da Imagem			Diferença no Tamanho	
Original	Criptografada	Decriptografada	Criptografada	Decriptografada
50.350	50.352	50.350	2	0
56.416	56.432	56.416	16	0
41.675	41.680	41.675	5	0
43.206	43.216	43.206	10	0
49.176	49.184	49.176	8	0
44.587	44.592	44.587	5	0
45.540	45.552	45.540	12	0
44.615	44.624	44.615	9	0
50.795	50.800	50.795	5	0
45.283	45.296	45.283	13	0

Nesta amostragem o pior caso foi um aumento de 16 **bytes**. Isto ocorre porque os tamanhos das imagens divergem e, se a quantidade de ***bits*** da imagem integrada com os ***bits*** da chave não totalizarem um número que seja divisível por 128 (tamanho do bloco utilizado pelo algoritmo), o último bloco de informação deve ser completado.

O mais importante nesta questão é que os ***bits*** que serviram para completar o último bloco foram descartados corretamente na função de decriptografia, não havendo nem perdas e nem ganhos de ***bits*** na imagem final.

O algoritmo AES demonstrou que não houve divergência no tamanhos das imagens criptografadas e decriptografadas na utilização das chaves de tamanho 128, 192 e 256 ***bits***, as três demonstraram o mesmo desempenho. Isto ocorre porque o tamanho do bloco utilizado pelo algoritmo é fixo e a quantidade de ***bits*** a serem "misturados" com a chave é superior ao tamanho dela em qualquer uma das três.

Os próximos capítulos demonstram os resultados obtidos para os testes de tempo decorrido de acordo com a quantidade de imagens criptografada/decriptografada e de acordo com o tamanho da chave.

4.2.1 Tempo Decorrido de Acordo com a Quantidade de Imagens

O tempo que o algoritmo AES, utilizando uma chave de 128 ***bits***, levou para criptografar uma imagem foi 0.10 segundos mais rápido do que o tempo gasto para decriptografar. No pacote de dez imagens a diferença foi mais significativa, sendo a criptografia mais lenta do que a decriptografia.

Com cinquenta imagens a função que decriptografa foi em torno de 0.16 segundos mais rápida do que a função que criptografa. Em um pacote de cem imagens o método de criptografia foi 0.36 segundos mais lento do que o método de decriptografia.

Já com duzentas imagens o tempo gasto com a decriptografia voltou a ser superior ao da criptografia, sendo a primeira 0.13 segundos mais lenta do que a segunda.

No melhor caso (criptografia de uma imagem) o algoritmo levou 0.023 segundos para realizar todo o processo e no pior caso (decriptografia do pacote de duzentas imagens) ocupou 1.304 segundos para finalizar a função.

O gráfico 2 demonstra a linha de crescimento do tempo (eixo vertical do gráfico) decorrido a partir da quantidade de imagens (eixo horizontal do gráfico) que são inseridas para o processo de criptografia/decriptografia.

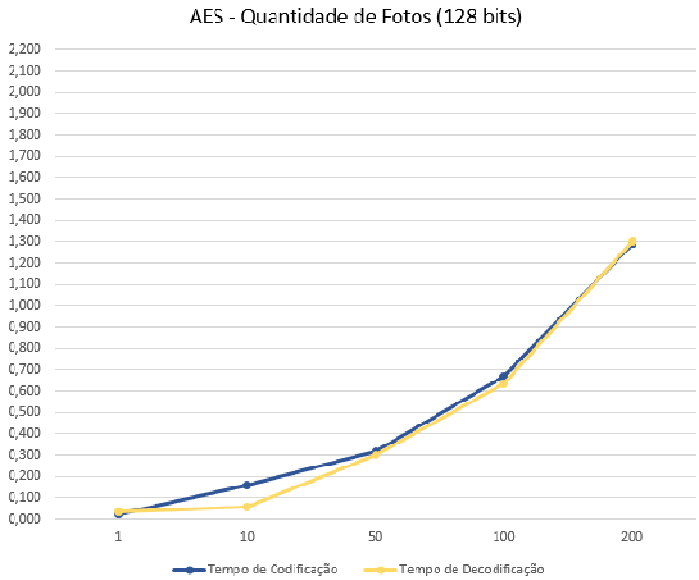

Gráfico 2 - Tempo do processo com AES utilizando chave de tamanho 128
Fonte: Autoria Própria

Quando alterado o tamanho da chave para 192 ***bits***, o tempo que a decriptografia ocupa ainda é superior ao da criptografia para uma única imagem. Já no pacote de dez imagens a decriptografia é 0.007 segundos mais rápida do que a criptografia.

Para os pacotes de cinquenta, cem e duzentas imagens a diferença entre os dois processos é mais significativa, sendo o tempo gasto com a criptografia superior à decriptografia no primeiro e no terceiro caso e a decriptografia superior à criptografia no segundo caso.

Neste conjunto de testes o melhor caso foi a criptografia de uma imagem (0.004 segundos) e o pior caso foi a criptografia de duzentas imagens (1.970 segundos).

O gráfico 3 demonstra o crescimento do tempo (eixo vertical), nota-se que o intervalo cresce linearmente somente no processo de codificação, no intervalo de cinquenta a duzentas imagens (eixo horizontal), nos demais intervalos o crescimento é não-linear.

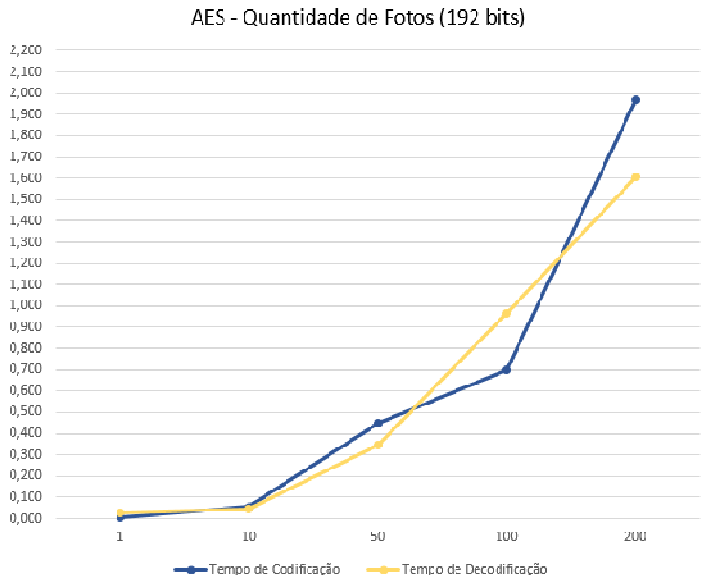

Gráfico 3 - Tempo do processo com AES utilizando chave de tamanho 192
Fonte: Autoria Própria

De acordo com os testes realizados utilizando o algoritmo AES com chave de 256 ***bits***, para uma imagem o tempo gasto nos dois processos foram praticamente

iguais, isto também ocorre no pacote com dez imagens. No pacote de cinquenta imagens o tempo gasto com a criptografia foi superior ao tempo da decriptografia.

As diferenças mais significativas de tempo neste teste foram com os pacotes de cem e duzentas imagens, onde a decriptografia foi 0.595 segundos mais lenta do que a criptografia no primeiro caso, e a criptografia ocupou tempo superior a decriptografia no segundo caso.

O pior caso do teste com chaves de 256 ***bits*** foi encontrado na criptografia de duzentas imagens ocupando 2.672 segundos para finalizar sua função, já o melhor caso foi a decriptografia de uma imagem – gastou 0.004 segundos para concluir o método.

O gráfico 4 ilustra os testes realizados de acordo com a quantidade de imagens (eixo horizontal). Nota-se que o crescimento do tempo (eixo vertical) foi não-linear neste caso.

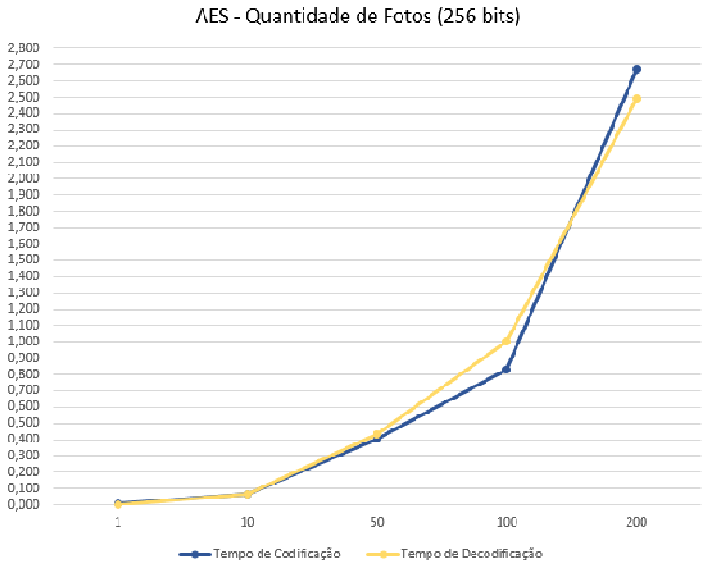

Gráfico 4 - Tempo do processo com AES utilizando chave de tamanho 256
Fonte: Autoria Própria

Analisando os três gráficos anteriores, é possível concluir rapidamente que é mais vantajoso realizar o processo imagem a imagem, pois assim o desempenho do algoritmo será superior. Mas, se os dados forem analisados minuciosamente, realizar o processo utilizando pacotes de imagens será mais vantajoso.

Multiplicado o tempo decorrido para a criptografia de uma imagem pela quantidade de imagens a serem criptografadas, tem-se uma base real do tempo que o algoritmo levaria, por exemplo: o melhor caso na utilização da chave de 128 ***bits*** ocupou 0.023s, multiplicado por 200 imagens (pacote máximo utilizado nos testes), resulta em 4.600s – este valor é quatro vezes maior do que o pior caso do mesmo teste (1.304s).

4.2.2 Tempo Decorrido de Acordo com o Tamanho da Chave

Como já foi explicado anteriormente, quanto maior o tamanho da chave, maior a segurança. Em teoria, quanto maior for a chave mais tempo o algoritmo gastaria para realizar os processos. Os gráficos desta seção demonstram os testes realizados para confirmar se isto se aplicaria a pratica na criptografia de grandes blocos (imagens).

Nota-se no gráfico 5 que para a criptografia de uma única imagem, a teoria não se aplica. Quanto maior o tamanho da chave, menor é o tempo decorrido para realização do processo.

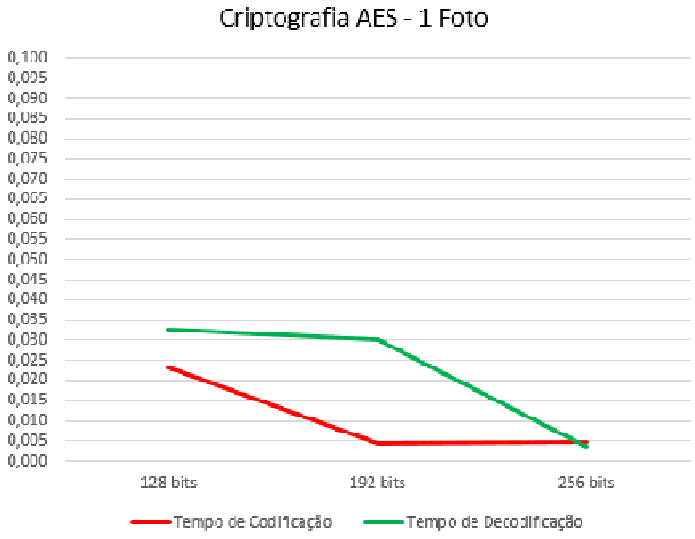

Gráfico 5 - Criptografia de uma imagem de acordo com o tamanho da chave
Fonte: Autoria Própria

É importante observar que tanto a chave de 192 ***bits*** quanto a de 256 ***bits*** apresentam o mesmo desempenho para a criptografia, mas na decriptografia a chave

de 192 ***bits*** ocupa um tempo superior. Já a chave de tamanho menor (128 ***bits***) foi a que apresentou o pior desempenho de tempo.

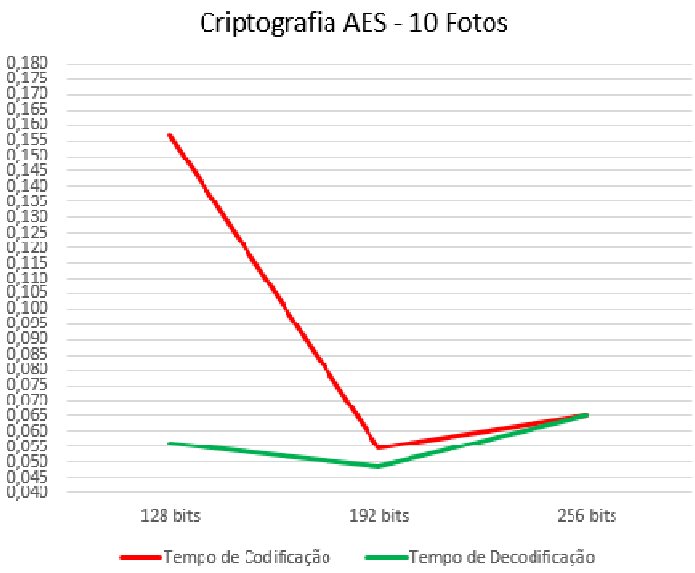

Gráfico 6 - Criptografia de dez imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

No processo de criptografia de dez imagens a teoria também não se aplica, mas, neste caso a chave intermediária (192 ***bits***) apresenta um melhor desempenho de tempo se comparada as outras. Os resultados são mostrados no gráfico 6.

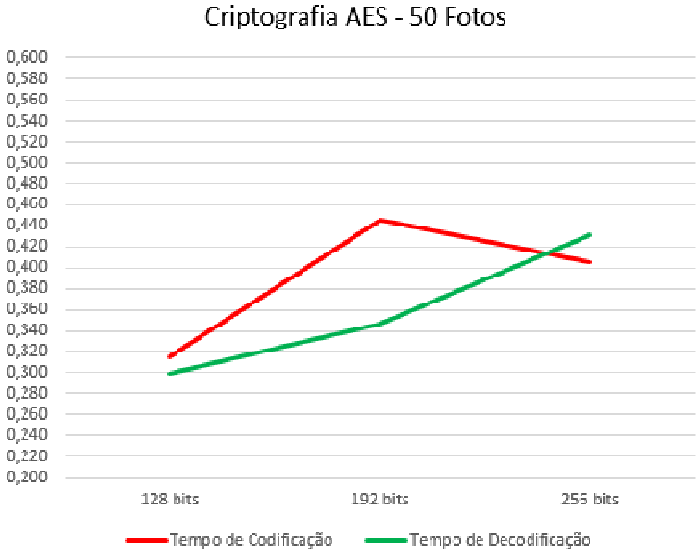

Gráfico 7 - Criptografia de cinquenta imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Já nos pacotes de cinquenta, cem e duzentas imagens a teoria se aplica. Nos três casos a menor chave apresentou o melhor desempenho de tempo tanto na criptografia quanto na decriptografia.

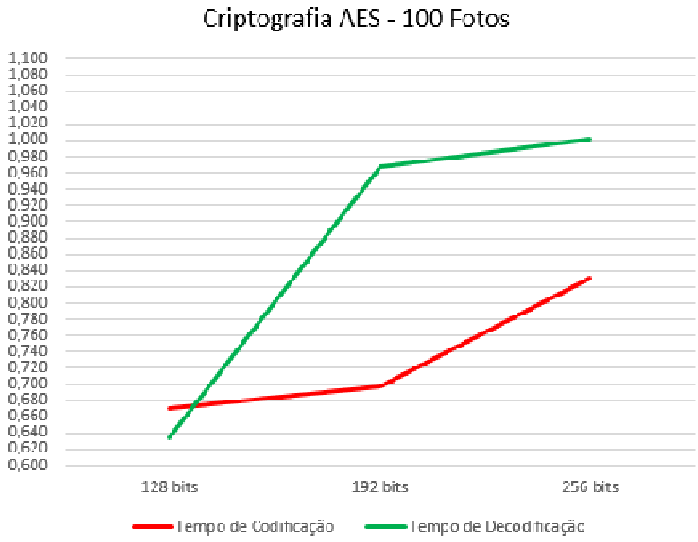

Gráfico 8 - Criptografia de cem imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

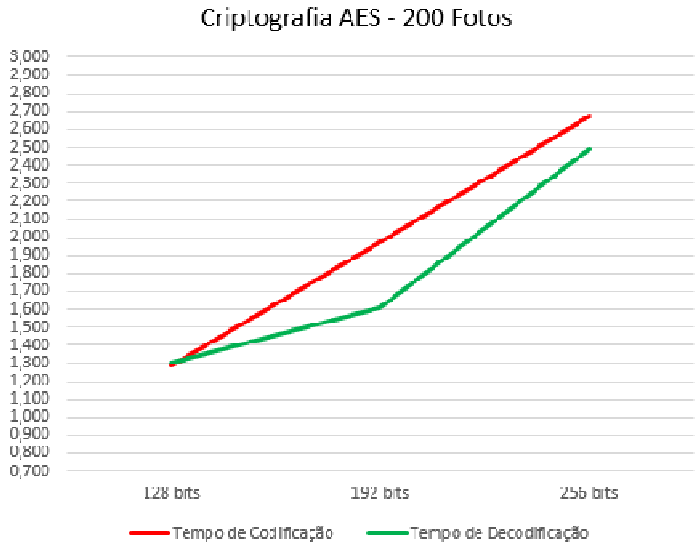

Gráfico 9 - Criptografia de duzentas imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Nos dois primeiros casos não houve um crescimento de tempo (eixo vertical) linear em nenhuma das duas funções. Já no último caso, a criptografia teve um

crescimento linear no tempo, dado o tamanho das chaves (eixo horizontal). Os três casos são apresentados nos gráficos sete, oito e nove.

Como os pacotes de imagem são de tamanhos variados e superiores ao tamanho da chave em qualquer uma das três, a chave deve ser utilizada mais de uma vez na sequência de ***bits***. Por isso que em alguns casos, chaves de tamanho superior apresentam maior desempenho, tudo depende do tamanho da imagem ou do pacote de imagens original.

Conforme o que foi descrito no capítulo 4.2.1, que é mais vantajoso criptografar pacotes de imagens ao invés de fazer o processo imagem-a-imagem, conclui-se que o algoritmo AES apresenta melhor desempenho na criptografia de dados simultâneos (pacotes) utilizando a chave de 128 ***bits***.

4.3 ALGORITMO BLOWFISH

Assim como no algoritmo anterior, nos testes realizados com o Blowfish foram observados: o tempo decorrido para criptografia/decriptografia de acordo com o tamanho das chaves e de acordo com a quantidade de imagens, o tamanho da imagem criptografada e o tamanho da imagem decriptografada.

O Blowfish apresentou um resultado satisfatório com relação ao tamanho das imagens após o termino do processo. A tabela abaixo demonstra os resultados obtidos em uma amostragem com dez imagens aleatórias.

Tabela 3 - Tamanho da imagem em *bytes* durante o processo com *Blowfish*

Tamanho da Imagem			**Diferença do Tamanho**	
Original	Criptografada	Decriptografada	Criptografada	Decriptografada
49.679	49.680	49.679	1	0
51.292	51.296	51.292	4	0
48.926	48.928	48.926	2	0
43.528	43.536	43.528	8	0
42.387	42.392	42.387	5	0
49.493	49.496	49.493	3	0
41.135	41.136	41.135	1	0
48.049	48.056	48.049	7	0
43.354	43.360	43.354	6	0
44.658	44.664	44.658	6	0

É possível notar que a diferença no tamanho da imagem original e da imagem decriptografada é zero, então, assim como o AES, o Blowfish não apresentou perda de dados.

Os resultados foram exatamente iguais para os três tamanhos de chaves, isto ocorre porque o tamanho do bloco utilizado pelo algoritmo é fixo. Não houve também uma grande diferença entre a imagem criptografada e a imagem original. Nota-se que o pior caso nesta amostragem foi de 8 ***bytes***.

Esta diferença ocorre por causa do ***Padding*** inserido na criação da cifra (ver capítulo 2.6.2), o qual completa o último bloco de dados caso o tamanho da informação (***bits*** da imagem original "misturados" aos ***bits*** da chave) não seja um número divisível por 64 (tamanho em ***bits*** do bloco utilizado pelo algoritmo).

O Blowfish demonstrou um resultado satisfatório nesse quesito, onde os ***bits*** utilizados para completar o bloco na função de criptografia foram descartados corretamente na função de decriptografia.

Os resultados de tempo decorrido de acordo com a quantidade de imagens e de acordo com o tamanho da chave são apresentados nos capítulos seguintes.

4.3.1 Tempo Decorrido de Acordo com a Quantidade de Imagens

Utilizando o algoritmo Blowfish com uma chave de tamanho 128 ***bits*** para criptografar/decriptografar uma única imagem foi possível observar que o tempo gasto com a codificação foi 0.12s superior ao da decodificação. No pacote de dez imagens o tempo de codificação também foi superior ao tempo de decodificação.

Nos pacotes de cinquenta, cem e duzentas imagens as diferenças foram mais significativas. No primeiro caso a codificação foi 0.246s mais rápida do que a decodificação, já no segundo caso a decodificação foi 0.113s mais rápida do que a codificação. No terceiro caso a codificação foi 0.554s mais lenta do que a decodificação.

A curva do tempo demonstra um crescimento não-linear a partir do aumento do tamanho dos pacotes. Logo, o pior caso (criptografia do pacote de duzentas imagens) levou cerca de 1.882s para realizar a sua função.

Já o melhor caso (decriptografia de uma única imagem) levou cerca de 0.014s para realizar a função. O gráfico 10 demonstra o intervalo de tempo (linha vertical) decorrido a partir da quantidade de imagens (linha horizontal).

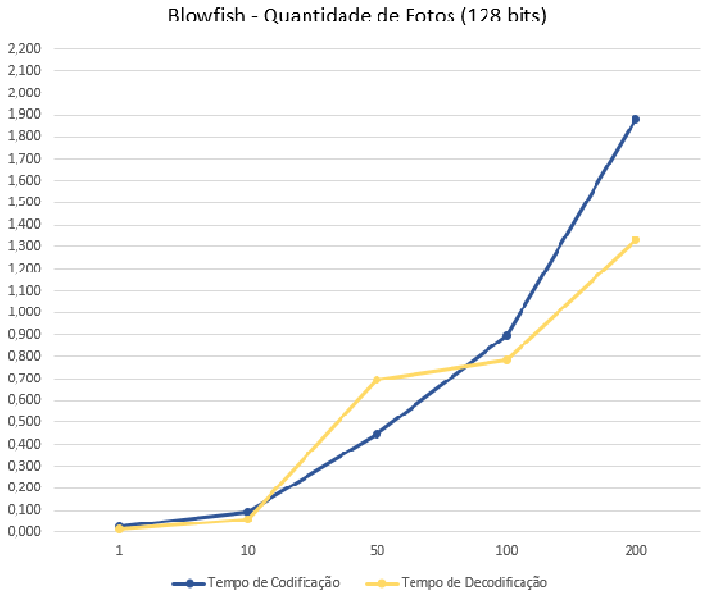

Gráfico 10 - Tempo decorrido com Blowfish e chave de tamanho 128 *bits*
Fonte: Autoria Própria

Nos testes realizados com a chave de tamanho 192 ***bits*** a codificação foi mais lenta do que a decodificação, tanto para uma única imagem quanto para o pacote de dez imagens.

Assim como na utilização da chave de tamanho menor, a diferença no tempo neste caso foi mais significativa nos pacotes de cinquenta, cem e duzentas imagens. No primeiro a codificação foi 0.408s mais lenta do que a decodificação.

Já nos dois últimos a decodificação foi mais lenta que a codificação, sendo que para cem imagens a diferença foi de 0.291s e para duzentas imagens a diferença foi de 0.400s.

O pior caso apresentado para este tamanho de chave foi a decodificação de duzentas imagens (1.956s) e o melhor caso foi a decodificação de uma única imagem (0.005s). O gráfico 11 ilustra a linha de tempo (eixo vertical) dado a quantidade de imagens (eixo horizontal).

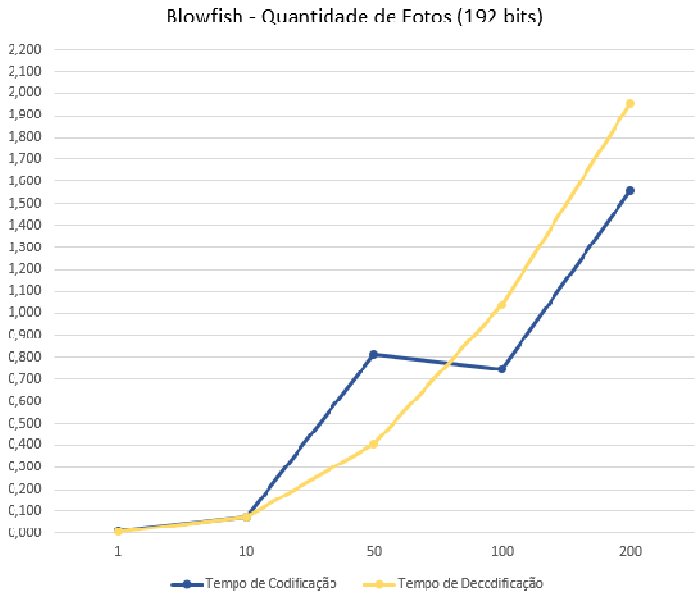

Gráfico 11 - Tempo decorrido com Blowfish e chave de tamanho 192 *bits*
Fonte: Autoria Própria

Os testes utilizando um tamanho de chave de 256 ***bits*** mostraram que o tempo decorrido para criptografar uma única imagem foi superior ao tempo para decriptografar. No pacote de dez imagens o tempo de decriptografia foi superior ao tempo de criptografia.

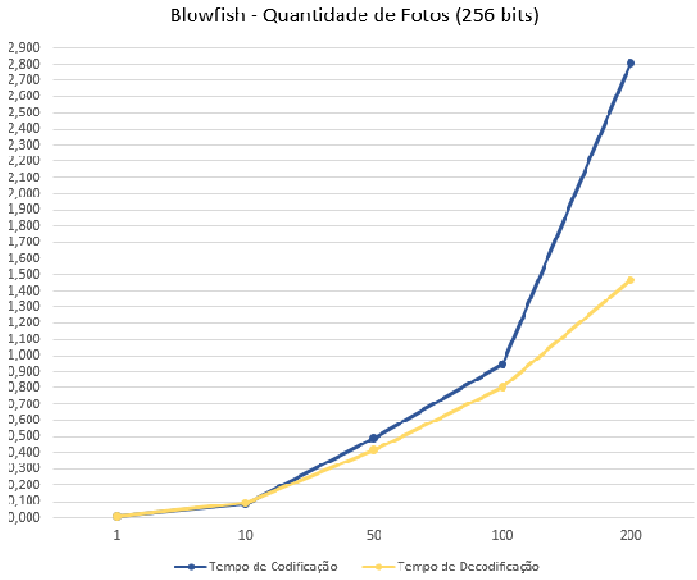

Gráfico 12 - Tempo decorrido com Blowfish e chave de tamanho 256 *bits*
Fonte: Autoria Própria

Nestes dois primeiros pacotes a diferença no tempo entre uma função e outra é pequena. Já nos outros três pacotes o tempo para criptografia foi maior que o tempo para decriptografia, havendo 0.075s de diferença entre uma função e outra no pacote de cinquenta imagens e 0.142s de diferença no pacote de cem imagens.

Já no pacote de duzentas imagens a diferença foi mais a significativa – 1.338s entre as duas funções. O gráfico 12 demonstra a curva no tempo decorrido dada a quantidade de imagens (eixo horizontal), nota-se que o salto maior se dá ao utilizar duzentas imagens. O melhor caso deste teste foi 0.005s (decodificação de uma única imagem) e o pior caso foi 2.807s (codificação de duzentas imagens).

Nota-se pelos gráficos que criptografar uma imagem acarreta um tempo menor, então, seria mais vantajoso criptografar várias imagens uma a uma e não em um pacote.

Para ter a certeza sobre este fato, foi multiplicado o tempo decorrido para a criptografia de uma única imagem em cada tamanho de chave, pela quantidade de imagens máxima dos testes (duzentas) e obteve-se os seguintes resultados:

Para a chave de 128 ***bits*** o tempo para criptografar as imagens seria de aproximadamente 5.200s e para a chave de 192 ***bits*** o tempo para criptografar imagem-a-imagem seria de aproximadamente 3.000s, então para estes dois casos o melhor desempenho se encontra em criptografar um pacote de imagens.

Já para a chave de 256 ***bits*** o tempo decorrido na criptografia seria aproximadamente 1.2s, então neste caso seria mais vantajoso criptografar imagem-a-imagem.

4.3.2 Tempo Decorrido de Acordo com o Tamanho da Chave

Existe a teoria de que para a criptografia de pequenas quantidades de dados, quanto maior for o tamanho das chaves, maior será o tempo decorrido para que o processo efetue todas as suas funções.

Esta condição foi testada na criptografia de grandes quantidades de ***bits*** (das imagens) utilizando o algoritmo Blowfish e os resultados são apresentados nos gráficos desta seção.

Para a criptografia de uma única imagem a teoria não se aplica, pois nota-se pelo gráfico 13 que a chave de tamanho maior (eixo horizontal) realizou todo o processo em um intervalo de tempo (eixo vertical) menor quando comparada às outras duas.

Nota-se também que para o processo de criptografia houve uma queda linear no tempo a partir do aumento do tamanho da chave.

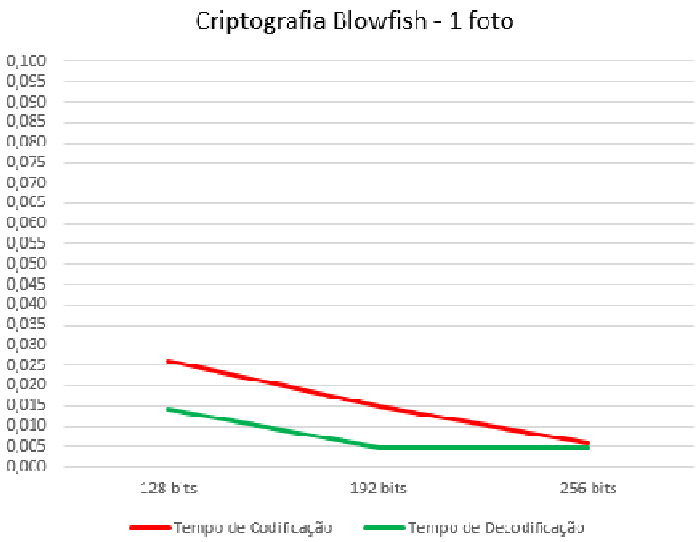

Gráfico 13 - Criptografia de uma imagem de acordo com o tamanho da chave
Fonte: Autoria Própria

Na criptografia de dez imagens a teoria se aplica somente à função de decriptografia, esta teve um crescimento linear do tempo a medida em que o tamanho da chave ia aumentando.

Já na criptografia a teoria não se aplica, pois o tempo decorrido utilizando a menor chave obteve um tempo maior do que as outras duas, sendo que para esta função a chave apresentou o melhor desempenho foi a de tamanho intermediário (192 ***bits***). O gráfico 14 ilustra esta condição.

Já o gráfico 15 demonstra o tempo decorrido (eixo vertical) no processo de criptografia de um pacote de cinquenta imagens dado o tamanho da chave (eixo horizontal).

A teoria neste caso se aplica a função de criptografia somente, onde a chave de menor tamanho obteve o melhor desempenho de tempo. Já na função de decriptografia

a teoria não se aplica, pois a chave que apresentou melhor desempenho foi a de tamanho 192 ***bits***.

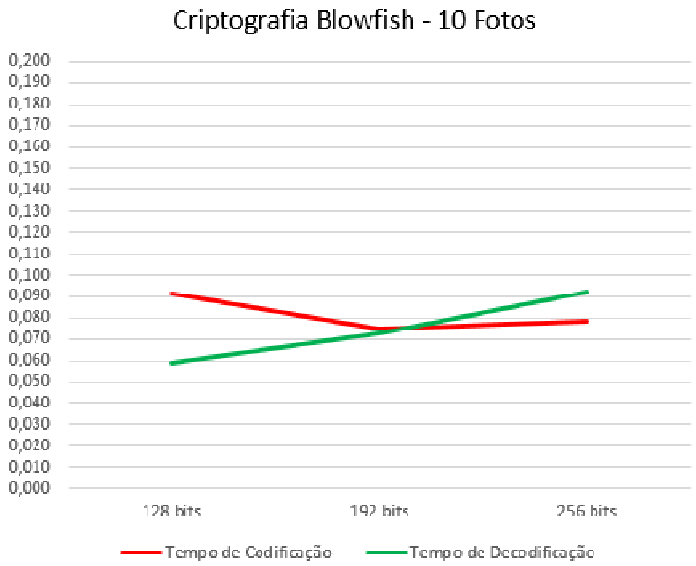

Gráfico 14 - Criptografia de dez imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

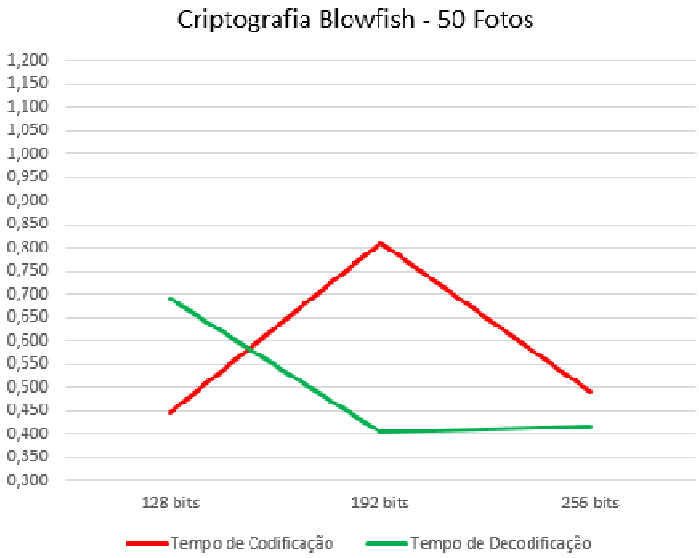

Gráfico 15 - Criptografia de cinquenta imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

A teoria, para os pacotes de cem e duzentas imagens, se aplica na função de decriptografia, onde a chave que apresenta melhor desempenho é a de tamanho

menor. Já na função de criptografia a chave intermediária (192 ***bits***) é a que tem o melhor desempenho de tempo.

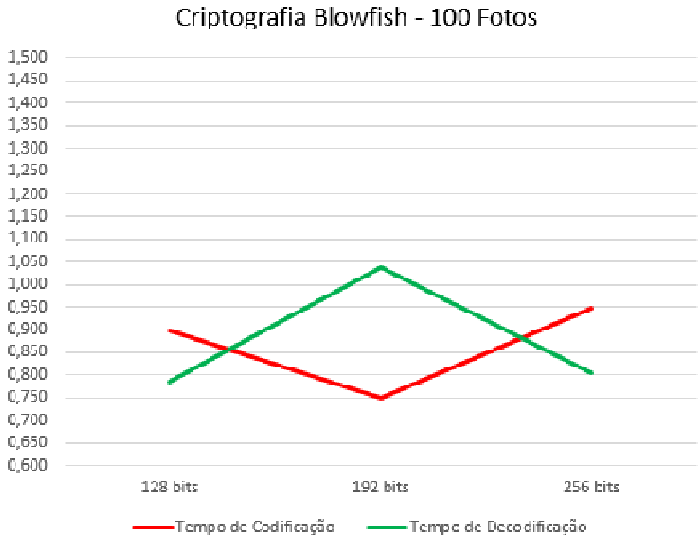

Gráfico 16 - Criptografia de cem imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

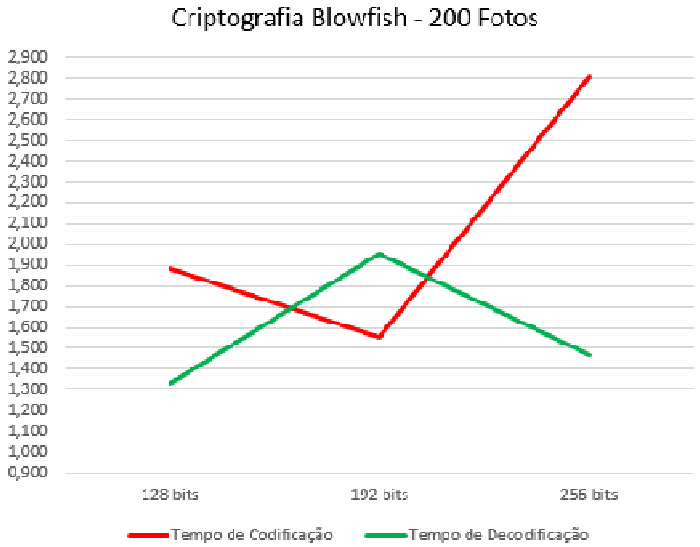

Gráfico 17 - Criptografia de duzentas imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Os gráficos 16 e 17 apresentam a linha do tempo decorrido (eixo vertical) para os pacotes de cem e duzentas imagens respectivamente, dado o tamanho das chaves (eixo horizontal).

Cada tamanho de chaves utilizada neste algoritmo teve suas particularidades. A chave de tamanho maior (256 ***bits***) só mostra um melhor desempenho no processo de criptografia de uma única imagem.

Tendo em vista o que foi elencado no capítulo 4.3.1 em que a melhor maneira de se criptografar com maior segurança é imagem-a-imagem pode-se concluir que este algoritmo pode ser melhor aproveitado em grandes quantidades de dados com a chave que possui maior segurança.

Nos demais pacotes de imagens não houve uma chave que obtivesse o melhor desempenho para o processo todo. Quando a chave de tamanho 128 ***bits*** se apresentava melhor para criptografia a chave de 192 ***bits*** se apresentava melhor para a decriptografia e vice-versa.

Sendo assim, não há como concluir qual chave se aplica melhor para cada quantidade (dez, cinquenta, cem e duzentas) de imagens.

4.4 ALGORITMO RC2

Os testes realizados nesta seção utilizando o algoritmo RC2 foram: tamanho da imagem criptografada, tamanho da imagem decriptografada, tempo decorrido de acordo com a quantidade de imagens e de acordo com o tamanho da chave.

O algoritmo RC2, assim como os dois primeiros algoritmos, apresentou resultados satisfatórios com relação ao tamanho das imagens criptografadas e decriptografadas.

O algoritmo apresentou valores exatamente iguais na utilização das três chaves (128, 192 e 256 ***bits***), isto porque para os testes o valor do bloco é fixado em 64 ***bits*** (conforme a descrição do algoritmo). Para demonstração dos tamanhos foi escolhida uma amostragem de dez imagens aleatórias, os resultados são apresentados na tabela abaixo.

Nota-se que não houve diferença entre o tamanho da imagem original e o tamanho da imagem decriptografada. Já na imagem criptografada o pior caso foi o acréscimo de 8 ***bytes***.

Tabela 4 - Tamanho da imagem em *bytes* durante o processo com RC2

Tamanho da Imagem			Diferença do Tamanho	
Original	Criptografada	Decriptografada	Criptografada	Decriptografada
42.696	42.704	42.696	8	0
49.596	49.600	49.596	4	0
47.741	47.744	47.741	3	0
54.036	54.040	54.036	4	0
41.596	41.600	41.596	4	0
42.730	42.736	42.730	6	0
48.894	48.896	48.894	2	0
47.654	47.656	47.654	2	0
42.078	42.080	42.078	2	0
50.002	50.008	50.002	6	0

Esta diferença no tamanho, encontrada na imagem codificada, se dá pelo fato de que, se o tamanho final da informação (***bits*** da imagem original concatenados com os ***bits*** da chave) não for um número múltiplo de 64 (tamanho em ***bits*** do bloco utilizado pelo RC2), a função deve inserir ***bits*** para completar o último bloco de informação cifrada.

O ponto interessante é que o algoritmo retirou totalmente os ***bits*** complementares na função de decriptografia, deixando a imagem final exatamente igual a imagem original.

Os testes de tempo (de acordo com o tamanho da chave e de acordo com a quantidade de imagens) são apresentados nos capítulos abaixo.

4.4.1 Tempo Decorrido de Acordo com a Quantidade de Imagens

Na realização dos testes com o algoritmo RC2 utilizando uma chave de 128 ***bits*** a função de decriptografia em todos os casos ocupou um tempo menor em comparação com a função de criptografia. Na criptografia de uma única imagem a diferença entre as duas funções foi de 0.016s, no pacote de dez imagens foi de 0.019s.

As diferenças mais significativas ocorreram a partir do pacote de cinquenta imagens, sendo neste caso de 0.106s. Já no pacote de cem imagens a diferença foi de 0.386s e de duzentas imagens foi de 0.754s.

O gráfico 18 mostra a linha do tempo (eixo vertical) decorrida de acordo com a quantidade de imagens (eixo horizontal). O pior caso deste se mostrou na criptografia

de duzentas imagens (2.186s) e o melhor caso se deu na decriptografia de uma única imagem (0.015s).

Nota-se que a partir do aumento na quantidade de imagens gerou-se uma curva de tempo, onde o maior salto se deu a partir do pacote de cem imagens.

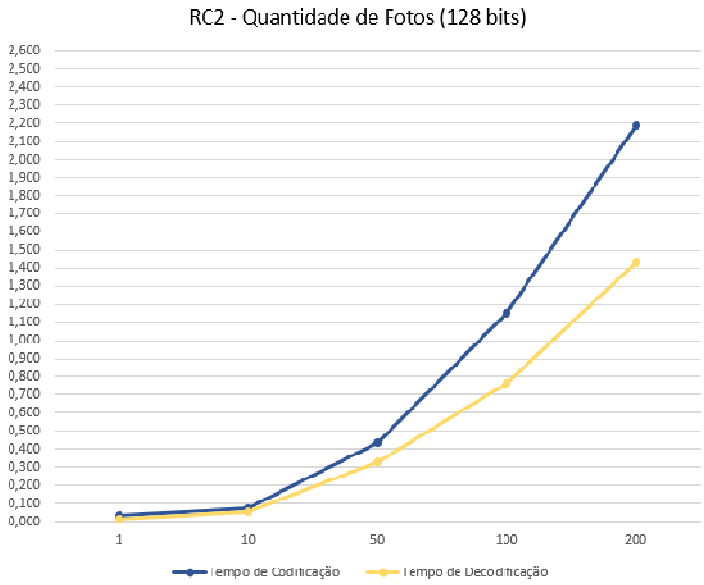

Gráfico 18 - Tempo decorrido com RC2 e chave de tamanho 128 *bits*
Fonte: Autoria Própria

Aumentando o tamanho da chave no processo para 192 ***bits***, as funções de criptografia deixaram de ocupar maior tempo em relação às funções de decriptografia em alguns casos. No processo com uma única imagem os tempos são iguais nas duas funções.

Já no pacote com dez imagens a função de criptografia foi 0.068s mais rápida do que a decriptografia. Com cinquenta imagens os dois processos ocuparam um intervalo de tempo parecido, mas a criptografia ocupou um tempo superior ao da decriptografia.

As diferenças mais significativas são na utilização de cem e duzentas imagens. No primeiro a decriptografia foi 0.589s mais lenta do que a criptografia, já no segundo a criptografia foi 0.620s mais lenta do que a decriptografia.

Nota-se pelo gráfico 19 que houve um salto considerável no tempo decorrido para a criptografia do pacote de duzentas imagens, já na função de decriptografia o maior salto foi registrado na utilização de cem imagens.

Logo, o melhor caso deste processo foi 0.005s (criptografia/decriptografia de uma única imagem) e o pior caso foi 2.177s (criptografia de duzentas imagens).

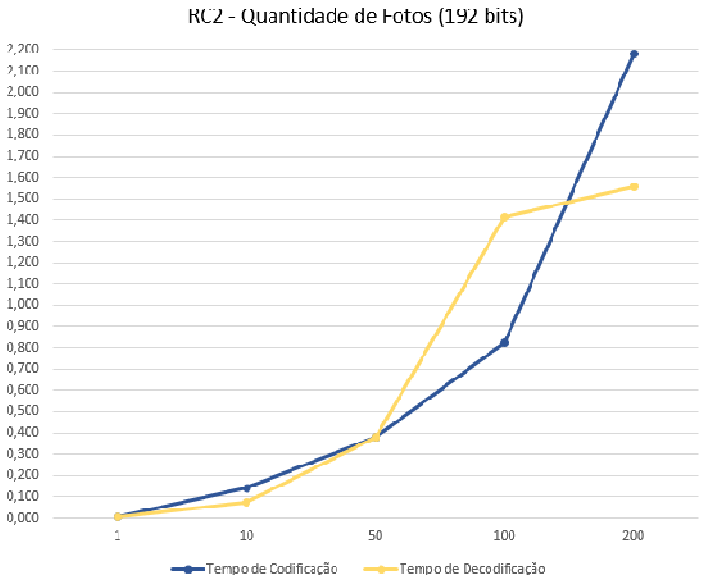

Gráfico 19 - Tempo decorrido com RC2 e chave de tamanho 192 *bits*
Fonte: Autoria Própria

Criptografando uma única imagem com o algoritmo RC2 e uma chave de 256 ***bits*** o tempo decorrido na função de decriptografia foi próximo ao tempo utilizado pela função de criptografia, mas ainda assim o tempo desta última foi superior. No pacote de dez imagens a função de criptografia também ocupou um tempo maior do que a função de decriptografia.

A criptografia de cinquenta imagens também foi superior à decriptografia da mesma quantidade. A diferença mais significativa foi na utilização de cem imagens, onde a criptografia foi 0.790s mais lenta do que a decriptografia. Já no pacote de duzentas imagens a decriptografia foi 0.099s mais lenta do que a criptografia.

O melhor caso deste processo foi 0.005s na decriptografia de uma única imagem, já o pior caso foi 1.610s na decriptografia de duzentas imagens. O gráfico 20

ilustra a linha do tempo (eixo vertical) decorrido a partir do aumento da quantidade de imagens (linha horizontal).

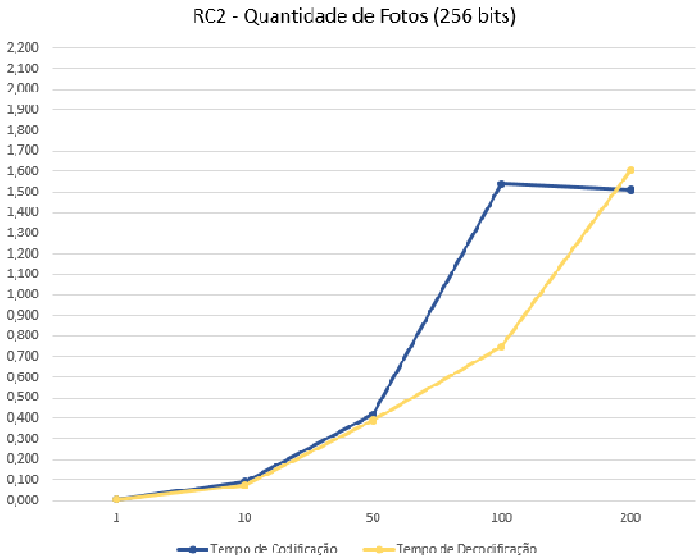

Gráfico 20 - Tempo decorrido com RC2 e chave de tamanho 256 *bits*
Fonte: Autoria Própria

Um dos objetivos deste trabalho é verificar qual é a melhor forma de se criptografar grandes quantidades de ***bits*** (imagens) – pacotes de imagens ou imagem-a-imagem.

Para concluir-se qual é a melhor forma, é necessário fazer a multiplicação do tempo decorrido para a criptografia de uma imagem pela quantidade máxima de imagens (duzentas) e comparar este resultado ao resultado obtido pela criptografia do pacote de duzentas imagens.

No processo realizado com a chave de 128 ***bits*** o tempo decorrido para a criptografia de uma imagem foi 0.031s, multiplicado pela quantidade máxima de imagens resultaria em um tempo de 6.200s – este valor é três vezes maior do que o utilizado para a criptografia de um pacote.

Com a chave de 192 ***bits*** o tempo decorrido para a criptografia de uma imagem foi de 0.005s, o que resultaria em um tempo de 1.000s – este valor é aproximadamente 1.000s menor do que o utilizado para a criptografia do pacote.

Com a chave de 256 ***bits*** o tempo decorrido na criptografia de uma única imagem foi 0.006s, resultando num tempo de 1.200s – este valor é 0.300s menor do que o tempo decorrido para criptografar o pacote de duzentas imagens.

Analisando estes dados conclui-se que só é vantajoso utilizar pacotes de duzentas fotos se a criptografia com RC2 for feita com uma chave de 128 ***bits***, nos outros casos é melhor realizar a criptografia imagem-a-imagem.

É importante frisar que estes valores são relativos, o tempo pode variar de acordo com a complexidade da chave criada e do poder de processamento do computador que irá realizar as funções.

4.4.2 Tempo Decorrido de Acordo com o Tamanho da Chave

Existe a teoria de que quanto maior a chave, maior o tempo decorrido para o processo. Um dos objetivos deste trabalho é testar esta informação – se inserida uma grande quantidade de ***bits*** (imagens) divididas em blocos de tamanho definido pelo algoritmo o tamanho da chave irá ditar o tempo decorrido no processo.

Os resultados deste teste utilizando o algoritmo RC2 são demonstrados nos gráficos deste capítulo.

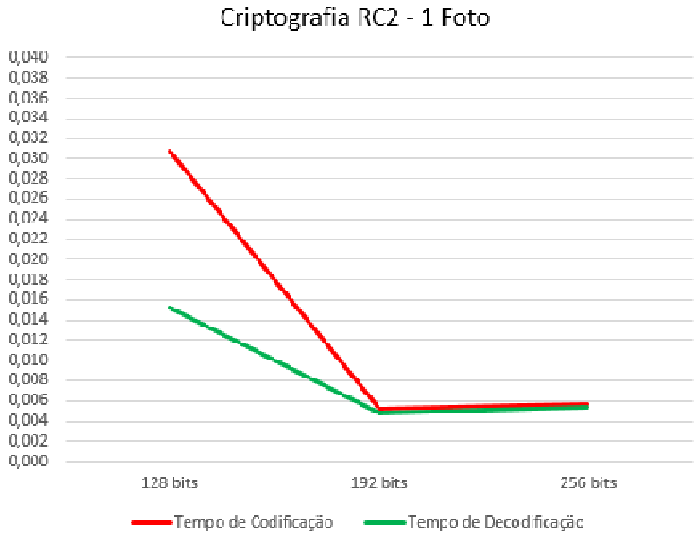

Gráfico 21 - Criptografia de uma imagem de acordo com o tamanho da chave
Fonte: Autoria Própria

Para a criptografia de uma única imagem esta teoria não se aplica, como pode ser observado no gráfico 21. A menor chave (128 ***bits***) foi a que apresentou o maior tempo decorrido para o processo.

Já as outras duas chaves ocuparam tempos praticamente iguais, sendo que a maior chave (256 ***bits***) diferenciou-se da outra em 0.001s na função de criptografia.

Já na criptografia do pacote de dez imagens a teoria se aplicou na chave de 128 ***bits*** de tamanho, ela obteve o melhor desempenho no tempo decorrido para o processo. Já a chave intermediaria (192 ***bits***) teve um desempenho inferior à chave de 256 ***bits*** na criptografia.

Na função de decriptografia a teoria se aplicou, pois a medida em que o tamanho da chave foi aumentando o tempo decorrido também aumentou. Estes resultados são ilustrados no gráfico 22.

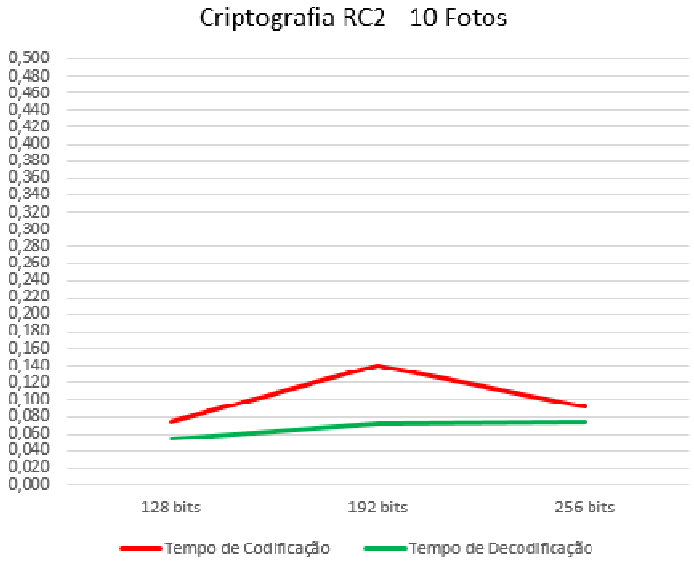

Gráfico 22 - Criptografia de dez imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Criptografando um pacote de cinquenta imagens com uma chave de 128 ***bits*** o tempo decorrido para a criptografia superou as outras duas, mas na função de decriptografia esta apresentou o melhor tempo.

A chave de 192 ***bits*** apresentou o melhor desempenho na função de criptografia e o tempo decorrido para a decriptografia, apesar de não ser o melhor, não teve grandes alterações.

Já a chave de 256 ***bits*** superou o tempo das outras na decriptografia e demonstrou um desempenho melhor sobre a menor chave e um desempenho inferior a chave intermediaria na função de criptografia. Os resultados deste teste são apresentados no gráfico 23.

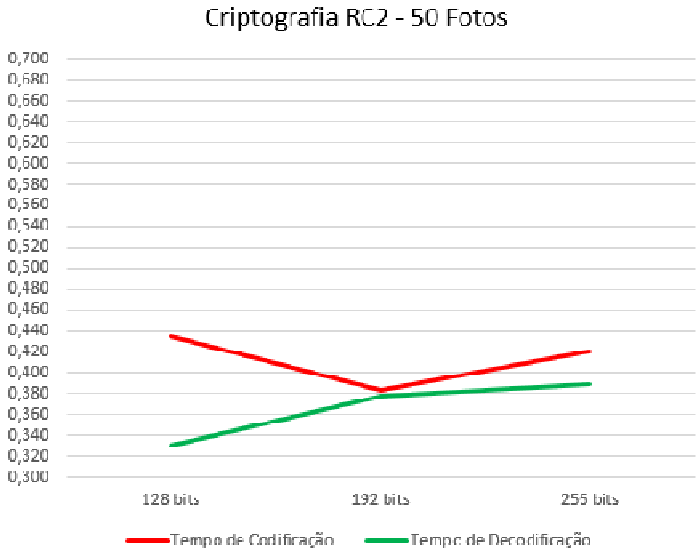

Gráfico 23 - Criptografia de cinquenta imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Na criptografia do pacote de cem imagens a teoria não se aplica. Nota-se pelo gráfico 24 que a chave de maior tamanho (256 ***bits***) foi a que mostrou melhor desempenho na função de decriptografia, mas foi a que ocupou o maior tempo na criptografia.

Já a chave de tamanho intermediário (192 ***bits***) foi a que mostrou melhor desempenho na criptografia, mas teve o pior desempenho na função de decriptografia. A chave de tamanho menor (128 ***bits***) não obteve nem o pior nem o melhor desempenho no processo.

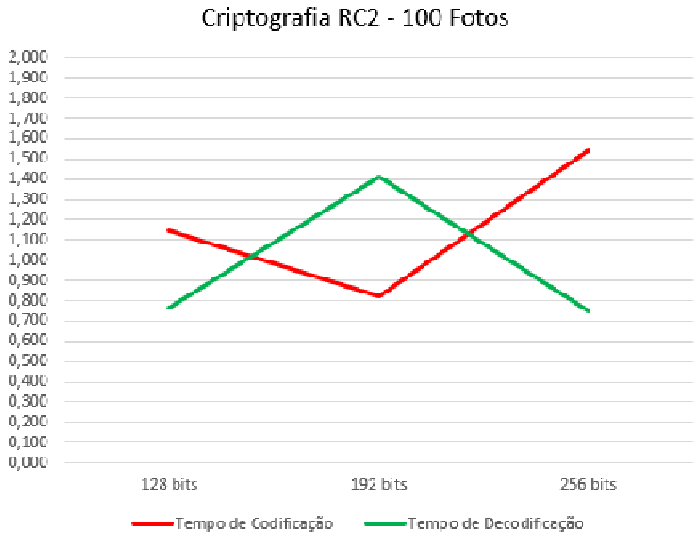

Gráfico 24 - Criptografia de cem imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Criptografando um pacote de duzentas imagens a teoria somente se aplica para a função de decriptografia, onde a chave de 128 ***bits*** foi a que melhor apresentou desempenho, mas foi a que mostrou o pior desempenho para a criptografia. A chave de 192 ***bits*** não apresentou nem o melhor, nem o pior desempenho para o processo.

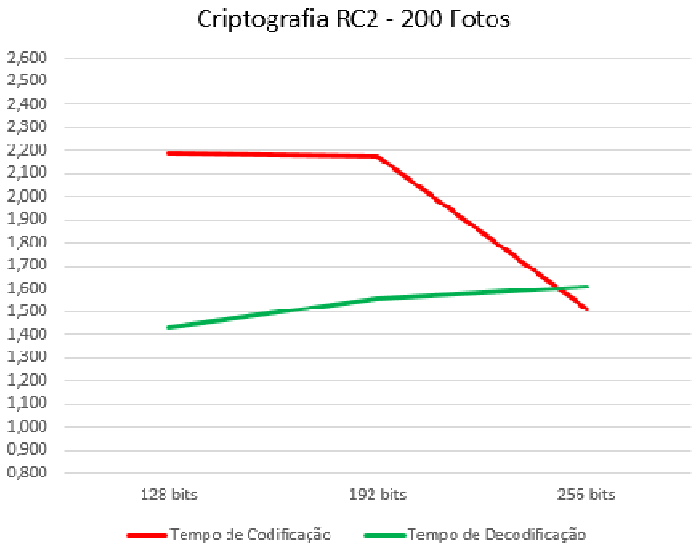

Gráfico 25 - Criptografia de duzentas imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Na função de criptografia a chave que ocupou o menor tempo foi a de 256 ***bits***, mas esta chave foi a que demonstrou o pior desempenho para a função de decriptografia. Os resultados deste teste são apresentados no gráfico 25.

O capítulo 4.4.1 conclui que é melhor utilizar uma criptografia com pacotes de duzentas imagens no processo com uma chave de 128 ***bits***, mas, os testes desta seção mostram que este tamanho de chave para a quantidade de imagens só apresenta vantagens na decriptografia, na criptografia ela apresenta o pior desempenho.

Já para os demais tamanhos de chave a seção anterior conclui que é mais vantajoso criptografar imagem-a-imagem. Os testes desta seção demonstraram que para a criptografia de uma única imagem quanto maior a chave melhor o desempenho.

Sendo assim, conclui-se que para o algoritmo RC2 é mais vantajoso criptografar uma quantidade x de imagens uma-a-uma utilizando a chave de tamanho 256 ***bits***.

4.5 ALGORITMO RC5

Os testes com o algoritmo RC5 foram realizados para comparar os tamanhos das imagens originais, criptografadas e decriptografadas, bem como o tempo decorrido de acordo com a quantidade de imagens utilizadas e de acordo com o tamanho da chave.

No teste do tamanho das imagens, o algoritmo RC5 apresentou um bom desempenho. Para ilustrar esses resultados foram escolhidas dez imagens aleatórias. Os resultados são apresentados na tabela 5.

Nota-se que não houve diferença entre o tamanho da imagem original com o tamanho da imagem decodificada. As diferenças são encontradas na comparação do tamanho da imagem original com o tamanho da imagem criptografada.

Nesta amostragem o pior caso apresentou-se no acréscimo de 7 ***bytes***. Esta diferença ocorre porque, se o tamanho da informação final não for um número múltiplo

de 64 (tamanho em ***bits*** do bloco conforme a especificação da API de criptografia) o algoritmo deve preencher o último bloco com ***bits*** aleatórios.

Tabela 5 - Tamanho da imagem em *bytes* durante o processo com RC5

Tamanho da Foto			**Diferença do Tamanho**	
Original	Criptografada	Decriptografada	Criptografada	Decriptografada
48.773	48.776	48.773	3	0
44.578	44.584	44.578	6	0
43.780	43.784	43.780	4	0
49.185	49.192	49.185	7	0
45.108	45.112	45.108	4	0
50.724	50.728	50.724	4	0
44.237	44.240	44.237	3	0
50.138	50.144	50.138	6	0
47.869	47.872	47.869	3	0
48.426	48.432	48.426	6	0

Tendo em vista isso, o RC5 – como todos os algoritmos testados neste trabalho – apresentou um resultado satisfatório na função de decriptografia, pois ele gerou a imagem final com o mesmo tamanho que a imagem original (não ocorreram perdas, nem ganhos de ***bytes***).

Os testes de tempo decorrido de acordo com o tamanho da chave e de acordo com a quantidade de imagens são descritos nas seções abaixo.

4.5.1 Tempo Decorrido de Acordo com a Quantidade de Imagens

Os testes realizados com o algoritmo RC5 neste capítulo demonstram a evolução do tempo decorrido para cada pacote de imagens dado o tamanho da chave utilizado.

Para uma chave de tamanho 128 ***bits*** criptografando uma, dez e cinquenta imagens o tempo decorrido nas duas funções do processo foram parecidos, mas a função de criptografia ocupou maior tempo com relação a decriptografia em todos os casos.

As diferenças mais significativas foram na criptografia de cem e duzentas imagens, sendo que a função de decriptografia utilizou tempo superior com relação a

criptografia. No primeiro caso a diferença foi de 0.090s e no segundo caso foi de 0.468s.

O gráfico 26 mostra o tempo decorrido (eixo vertical) conforme o aumento da quantidade de imagens (eixo horizontal), nota-se que o maior salto se deu na criptografia de duzentas imagens. O melhor caso deste processo foi na decriptografia de uma única imagem (0.012s), já o pior caso foi na decriptografia de duzentas imagens (1.496).

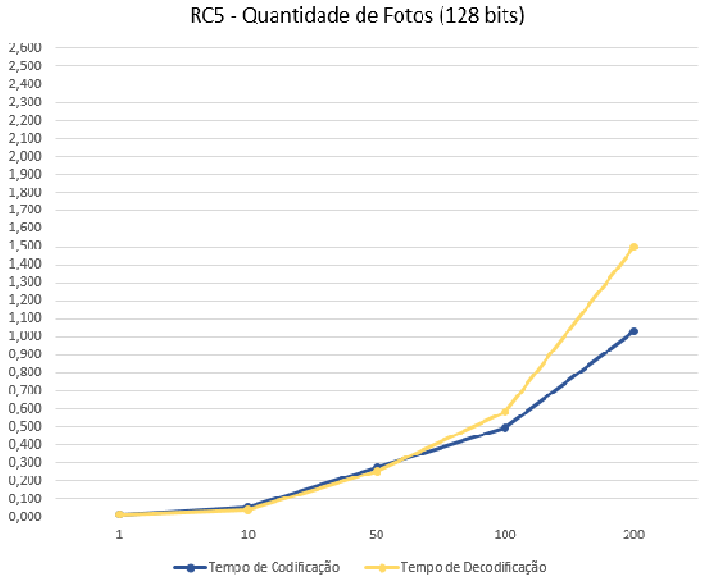

Gráfico 26 - Tempo decorrido com RC5 e chave de tamanho 128 *bits*
Fonte: Autoria Própria

Utilizando uma chave de 192 ***bits*** para a criptografia de uma, dez, cinquenta, cem e duzentas imagens, a função de decriptografia demonstrou um melhor desempenho no tempo comparada a criptografia em todos os casos. Em uma única imagem e no pacote de dez imagens a diferença entre as duas funções foi pequena.

Já nos demais pacotes a diferença foi significativa. Para cinquenta imagens foi de 0.073s, com cem imagens foi de 0.410s e para duzentas imagens foi de 0.174s. O maior salto demonstrado neste processo foi com a utilização do pacote de cem imagens.

O pior caso apresentado neste teste foi de 1.318s (criptografia do pacote de duzentas imagens), já o melhor caso foi 0.004s (decriptografia de uma única imagem).

O gráfico 27 demonstra os resultados do tempo decorrido (eixo vertical) de acordo com o aumento no tamanho dos pacotes de imagens (eixo horizontal).

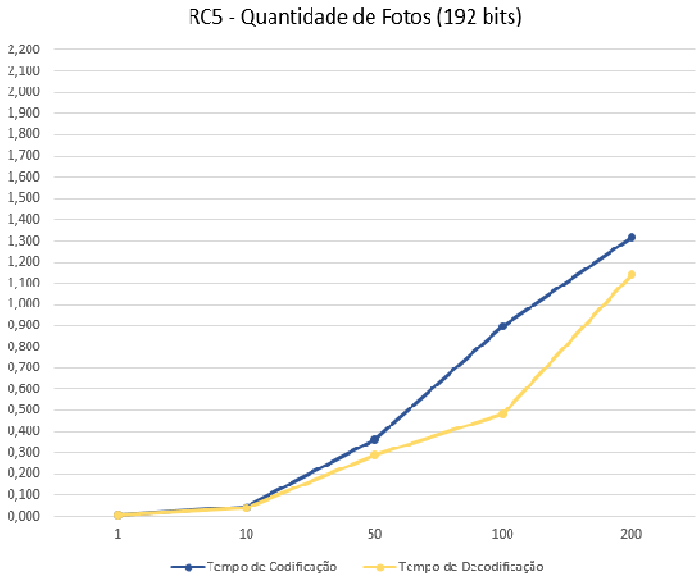

Gráfico 27 - Tempo decorrido com RC5 e chave de tamanho 192 *bits*
Fonte: Autoria Própria

O processo de criptografia utilizando uma chave de tamanho 256 ***bits*** demonstra que para uma única imagem e para um pacote de dez imagens os tempos decorridos são parecidos nas duas funções, mas a decriptografia apresenta o melhor desempenho se comparada a criptografia.

No pacote de cinquenta imagens a decriptografia apresentou um desempenho inferior à criptografia, sendo a diferença entre as duas de 0.087s. Nos pacotes de cem e duzentas imagens a decriptografia volta a apresentar um desempenho melhor do que a criptografia.

O maior salto apresentado neste processo foi com a utilização do pacote de cem imagens, logo a diferença entre as duas funções foi de 0.487s. Na utilização das duzentas imagens a diferença foi de 0.096s.

O gráfico 28 demonstra os resultados apresentados neste teste a partir do tempo decorrido (eixo vertical) de acordo com o acréscimo da quantidade de imagens (eixo horizontal). O melhor caso deste teste foi a decriptografia de uma única imagem (0.004s) e o pior caso foi a criptografia do pacote de duzentas imagens (1.494s).

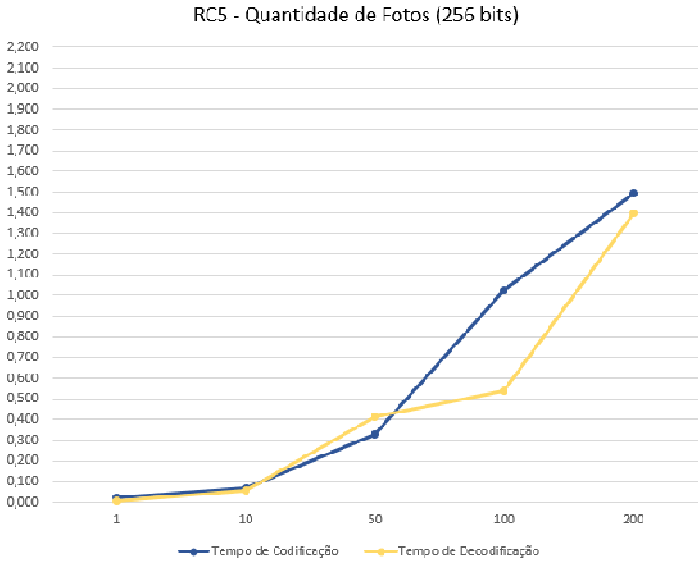

Gráfico 28 - Tempo decorrido com RC5 e chave de tamanho 256 *bits*
Fonte: Autoria Própria

Nota-se pelos gráficos que a criptografia de uma única imagem ocupou um tempo bem menor se comparada a criptografia do pacote de duzentas imagens. Mas para concluir qual é a melhor maneira de se criptografar uma quantidade x de imagens, será exemplificado o tempo decorrido de uma possível criptografia imagem-a-imagem.

Para isso é demonstrado abaixo a multiplicação dos valores obtidos na criptografia de uma única imagem pela quantidade máxima de imagens (duzentas) utilizada nos testes.

Na criptografia com a chave de 128 ***bits*** o tempo decorrido foi de 0.014s (para uma única foto). Neste caso é mais vantajoso criptografar um pacote de duzentas imagens – pois ele ocupou um tempo de 1.028s – logo, se criptografar imagem-a-imagem o tempo decorrido seria de 2.800s.

Utilizando a chave de 192 ***bits*** o tempo decorrido para criptografar uma única imagem foi de 0.005s. Neste caso a maior vantagem seria o processo imagem-a-imagem (ocuparia 1.000s), pois para criptografar o pacote de duzentas imagens o algoritmo ocupou um tempo de 1.318s.

Com a chave de 256 ***bits*** a criptografia de uma única imagem ocupou um tempo de 0.023s, para esta a vantagem é a criptografia de um pacote de duzentas

imagens (ocupou um tempo de 1.494s), logo utilizada a criptografia imagem-a-imagem o tempo decorrido seria de 4.600s.

Assim conclui-se que a única chave que apresenta vantagem na criptografia imagem-a-imagem é a de tamanho 192 ***bits***, as outras duas apresentam o melhor desempenho com pacotes de grandes quantidades de imagens.

4.5.2 Tempo Decorrido de Acordo com o Tamanho da Chave

Um dos objetivos deste trabalho é verificar se a teoria da criptografia de texto se aplica à criptografia de grandes quantidades de ***bits*** (neste caso específico: as imagens). A teoria acima citada diz que quanto maior for a chave, maior será o tempo decorrido para finalizar o processo.

Os gráficos desta seção demonstram os resultados para estes testes de tempo decorrido (eixo vertical) de acordo com o acréscimo no tamanho das chaves (eixo horizontal) utilizando o algoritmo RC5.

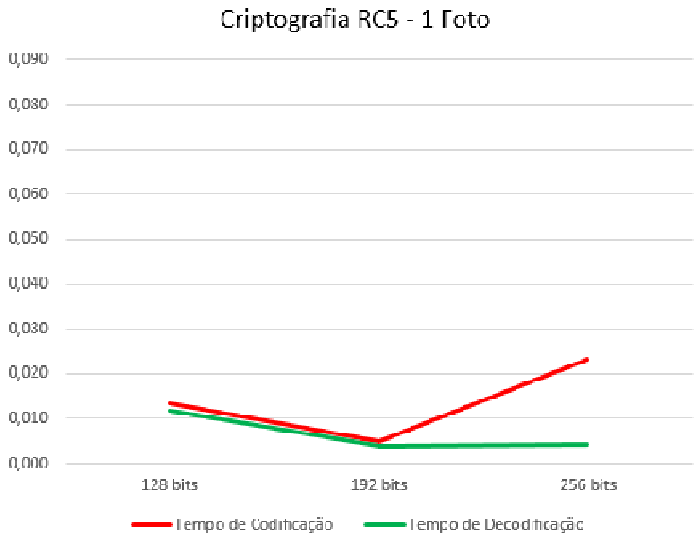

Gráfico 29 - Criptografia de uma imagem de acordo com o tamanho da chave
Fonte: Autoria Própria

Na criptografia de uma única imagem a teoria não se aplica. Na função de criptografia a chave que apresentou o melhor desempenho foi a de tamanho

intermediário (192 ***bits***), seguida da chave de 128 ***bits***. A chave que apresentou o pior desempenho nesta função foi a de tamanho maior (256 ***bits***).

Na função de decriptografia as chaves de tamanhos maiores (192 e 256 ***bits***) apresentaram o mesmo desempenho de tempo decorrido, logo, a chave de tamanho menor apresentou o pior desempenho. Os resultados deste teste são apresentados no gráfico 29.

Para a criptografia do pacote de dez imagens a teoria também não se aplica, pois a chave que apresentou melhor desempenho no processo foi a de tamanho 192 ***bits***, seguida da chave de tamanho 128 ***bits***. A chave que apresentou o pior desempenho foi a de tamanho 256 ***bits***. O gráfico 30 ilustra estes resultados.

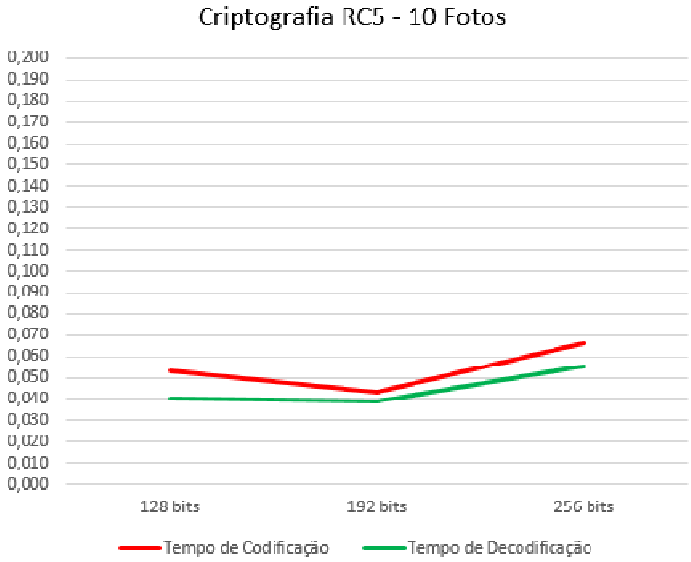

Gráfico 30 - Criptografia de dez imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Já na criptografia de cinquenta imagens a teoria se aplica. A chave que apresentou o melhor desempenho no processo foi a de tamanho menor (128 ***bits***). Para a função de criptografia a chave que apresentou o pior desempenho foi a de tamanho 192 ***bits***.

Já para a função de decriptografia a chave de 256 ***bits*** foi a que apresentou o pior desempenho. Os resultados são apresentados no gráfico 31.

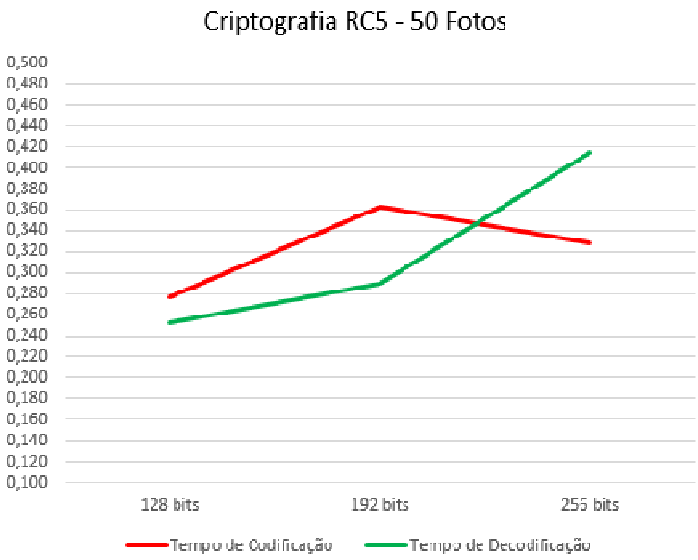

Gráfico 31 - Criptografia de cinquenta imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Criptografando um pacote de cem imagens a teoria somente se aplica na função de criptografia, onde a chave de menor tamanho (128 ***bits***) apresenta o melhor desempenho, seguida da chave de 192 ***bits***.

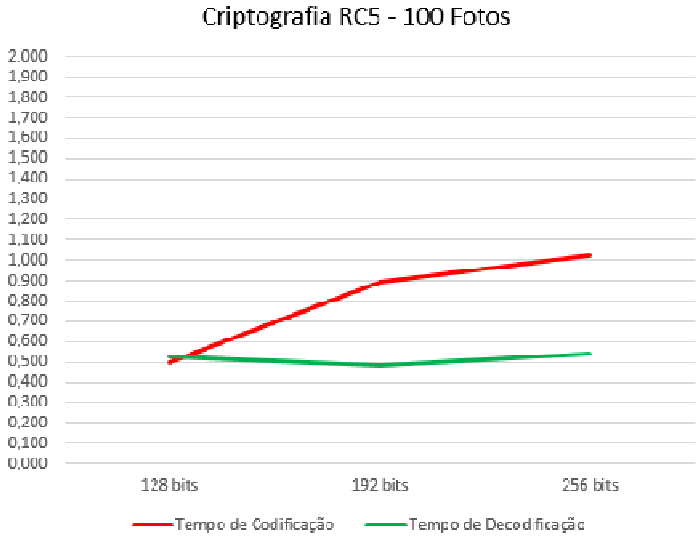

Gráfico 32 - Criptografia de cem imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Na função de decriptografia a chave que apresenta o melhor desempenho é a de 192 ***bits***, seguida da chave de 128 ***bits***. Já a chave de 256 ***bits*** foi a que apresentou

o pior desempenho no processo. O gráfico 32 demonstra os resultados obtidos neste teste.

O gráfico 33 demonstra os resultados obtidos na criptografia de um pacote de duzentas imagens, onde a teoria somente se aplica na função de criptografia, a chave de menor tamanho (128 ***bits***) é a que apresenta o melhor desempenho, seguida da chave de 192 ***bits***.

A chave de 256 ***bits*** para esta função apresenta o pior desempenho. Na função de decriptografia a chave que apresentou o melhor desempenho foi a de tamanho intermediário (192 ***bits***), seguida da chave de tamanho 256 ***bits***. A chave que apresentou o pior desempenho para esta função foi a de tamanho menor (128 ***bits***).

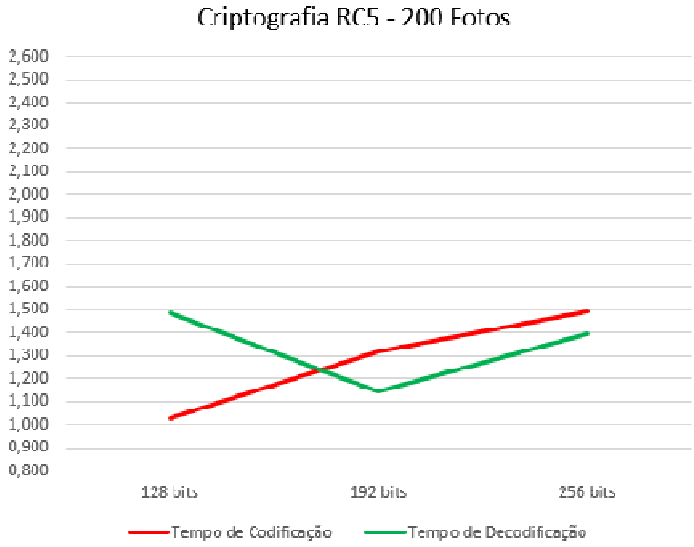

Gráfico 33 - Criptografia de duzentas imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

A conclusão do capítulo anterior foi que a chave de tamanho 192 ***bits*** apresenta sua melhor vantagem na criptografia imagem-a-imagem. Mostrou-se nesta seção que para este tipo de processo esta chave apresenta o melhor desempenho. Então, se a chave escolhida para uma criptografia de X imagens for a de tamanho intermediário conclui-se que o melhor método é a criptografia imagem-a-imagem.

Conclui-se para os dois outros tamanhos de chave que o melhor método é a criptografia de um pacote de imagens, mas, nesta seção o resultado demonstra que cada chave possui uma vantagem para cada função, então não se pode precisar qual e

a melhor chave para a criptografia desta quantidade de imagens no quesito tempo decorrido.

Apesar de não ser testado a questão da segurança das chaves neste trabalho, a literatura explica que quanto maior a chave, maior a segurança, então para um pacote de X imagens o melhor método ainda se encontra na utilização da chave de 256 ***bits***.

4.6 ALGORITMO RC4

O algoritmo RC4 foi utilizado neste trabalho para comparar os tempos decorridos por ele para o processo de acordo com a quantidade de imagens e de acordo com o tamanho da chave.

Foi utilizado também para comparar o tamanho da imagem criptografada e o tamanho da imagem decriptografada. De todos os algoritmos verificados neste trabalho o RC4 foi o que apresentou os melhores resultados para este quesito.

Assim como os demais algoritmos, os tamanhos dos arquivos criptografados e decriptografados foram exatamente iguais para os três tamanhos de chave utilizadas nos testes (128, 192 e 256 ***bits***).

Para ilustrar os dados obtidos com este algoritmo foram selecionadas dez imagens aleatórias, as quais são demonstradas na tabela abaixo:

Tabela 6 - Tamanho da imagem em *bytes* durante o processo com RC4

Tamanho da Foto			**Diferença do Tamanho**	
Original	Criptografada	Decriptografada	Criptografada	Decriptografada
45.470	45.470	45.470	0	0
44.955	44.955	44.955	0	0
49.741	49.741	49.741	0	0
43.819	43.819	43.819	0	0
49.181	49.181	49.181	0	0
53.474	53.474	53.474	0	0
49.255	49.255	49.255	0	0
53.346	53.346	53.346	0	0
49.693	49.693	49.693	0	0
53.217	53.217	53.217	0	0

Nota-se que não houve nenhum acréscimo no tamanho das imagens criptografadas e decriptografadas. Isto ocorre porque o algoritmo não utiliza-se de blocos para ciframento e sim de um fluxo continuo – no momento em que os ***bits*** da imagem original são “misturados” aos ***bits*** da chave eles são gravados no arquivo. Não há perdas nem ganhos de ***bytes*** nas imagens.

Nos capítulos abaixo são apresentados os resultados de tempo decorrido de acordo com a quantidade de imagens e de acordo com o tamanho da chave.

4.6.1 Tempo Decorrido de Acordo com a Quantidade de Imagens

Os testes realizados nesta seção demonstram a linha do tempo decorrido a partir do acréscimo da quantidade de imagens para cada tamanho de chave. Utilizando uma chave de 128 ***bits*** na criptografia de uma e dez imagens o tempo decorrido para a função de decriptografia obteve um melhor desempenho em comparação a criptografia.

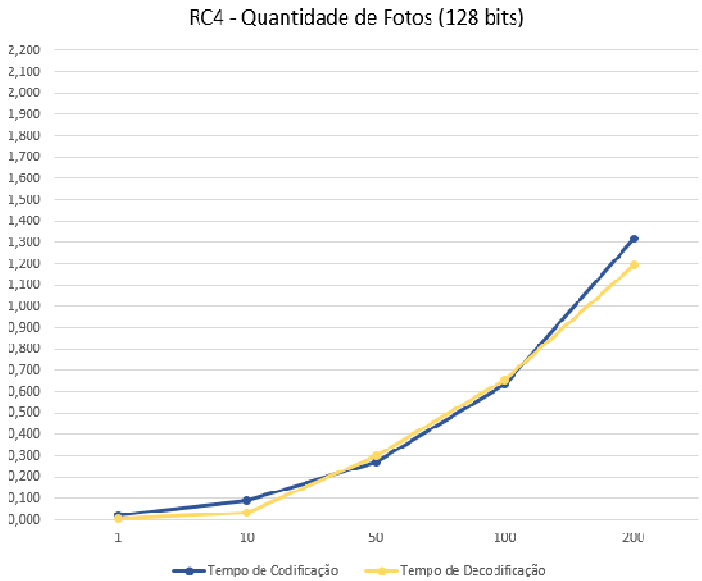

Gráfico 34 - Tempo decorrido com RC4 e chave de tamanho 128 *bits*
Fonte: Autoria Própria

Com dez imagens a diferença entre as duas funções foi de 0.054s. Já nos pacotes de cinquenta e cem imagens a função de criptografia obteve um melhor desempenho em relação a função de decriptografia. A diferença mais significativa deste

teste apresentou-se na criptografia de duzentas imagens, onde a função de criptografia foi 0.121s mais lenta do que a decriptografia.

O melhor caso deste foi apresentado na decodificação de uma única imagem (0.006s), já o pior caso foi na criptografia de duzentas imagens (1.318s). O gráfico 34 demonstra a linha do tempo decorrido (eixo vertical) de acordo com a quantidade de imagens (eixo horizontal).

Na utilização de uma chave de tamanho 192 ***bits*** para criptografar uma, dez, cinquenta, cem e duzentas imagens, a função de decriptografia apresentou um melhor desempenho sobre a criptografia em todos os casos.

Nos dois primeiros, os tempos decorridos para cada função são parecidos, a diferença começa a ser significativa a partir do pacote de cinquenta imagens, neste ela foi de 0.112s. Com cem imagens a diferença foi de 0.258s, e com duzentas imagens foi de 0.283s.

O gráfico 35 demonstra os resultados obtidos neste teste. O melhor caso foi 0.004s na decriptografia de uma única imagem e o pior caso foi 1.540s na criptografia de duzentas imagens.

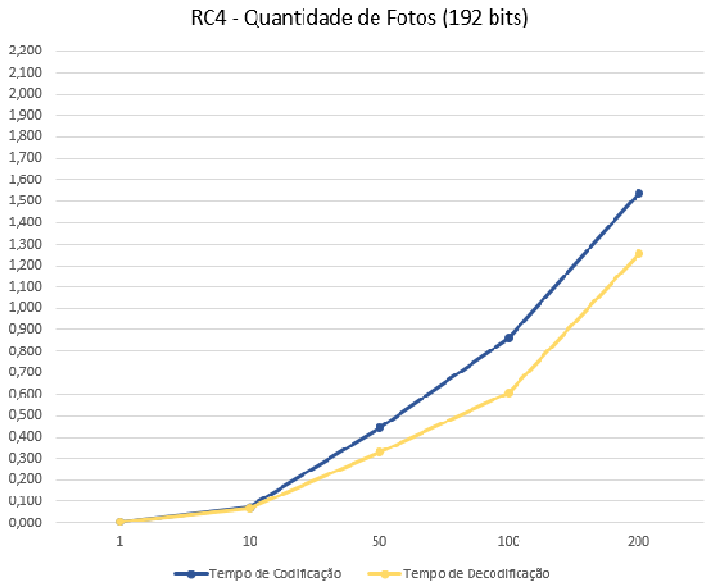

Gráfico 35 - Tempo decorrido com RC4 e chave de tamanho 192 *bits*
Fonte: Autoria Própria

Assim como na utilização da chave de 192 ***bits***, a chave de 256 ***bits*** na função de decriptografia apresentou um melhor desempenho sobre a criptografia para todas as quantidades de imagens. Com uma e dez imagens as duas funções decorreram tempos parecidos.

As diferenças mais significativas são apresentadas a partir do processo com cinquenta imagens, nesta foi de 0.304s. Com cem imagens a diferença foi de 1.640s e com duzentas imagens foi de 0.714s. O maior salto se deu com a utilização do pacote de cem imagens.

O pior caso deste teste foi encontrado na criptografia de duzentas imagens (2.715s) e o melhor caso foi a decriptografia de uma única imagem (0.019s). O gráfico 36 demonstra a linha do tempo decorrido (eixo vertical) a partir do aumento da quantidade de imagens (eixo horizontal).

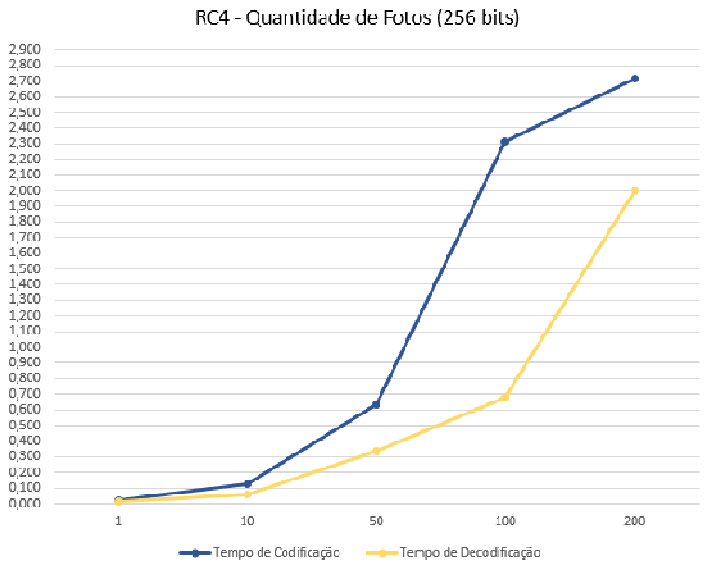

Gráfico 36 - Tempo decorrido com RC4 e chave de tamanho 256 *bits*
Fonte: Autoria Própria

De acordo com os gráficos a diferença entre a criptografia de uma única imagem e o pacote de duzentas imagens é bem significativa. Para concluir se o melhor método para codificar uma quantidade X de imagens é utilizando pacotes ou imagem-a-imagem é necessário multiplicar o tempo decorrido para a codificação de uma imagem pela quantidade de imagens.

Para exemplificar esta multiplicação será feita com a quantidade máxima de imagens utilizadas nos testes (duzentas) e o resultado comparado com o tempo decorrido para criptografia do pacote de imagens.

Utilizando uma chave de tamanho 128 ***bits*** o tempo decorrido para criptografar uma única imagem foi de 0.024s. Então conclui-se que para este tamanho de chave é mais vantajoso criptografar um pacote de duzentas imagens (o tempo gasto foi de 1.318s) do que codificar uma a uma (o tempo gasto seria de 4.800s).

Para uma chave de tamanho 192 ***bits*** o tempo decorrido para codificar uma imagem foi de 0.005s. Neste tamanho de chave o tempo que seria gasto para criptografar uma a uma seria de 1.000s, conclui-se assim que este seria o melhor método, tendo em vista que a criptografia de um pacote de duzentas imagens gastou 1.540s.

A chave de 256 ***bits*** de tamanho ocupou um tempo de 0.023s para criptografar uma única imagem. Logo, conclui-se que para este caso é mais vantajoso codificar um pacote de duzentas imagens (o tempo gasto foi de 2.715s) do que criptografar uma imagem por vez (seria gasto 4.200s para concluir a função).

4.6.2 Tempo Decorrido de Acordo com o Tamanho da Chave

Há a teoria de que quanto maior for a chave maior será o tempo decorrido para a criptografia do texto puro. Esta seção testa esta informação na utilização de uma grande quantidade de ***bits*** (imagens).

Os resultados são apresentados nos gráficos a seguir – onde o eixo vertical apresenta o intervalo de tempo decorrido e o eixo horizontal apresenta a quantidade de imagens utilizada.

Para a criptografia de uma única imagem a teoria não se aplica. Neste caso a chave que apresentou o melhor desempenho no tempo foi a de tamanho 192 ***bits***. Na função de codificação a chave que apresentou o pior desempenho de tempo foi a de menor tamanho (128 ***bits***).

Já na função de decriptografia a chave que apresentou o pior desempenho foi a de tamanho 256 ***bits***. O gráfico 37 ilustra estes resultados.

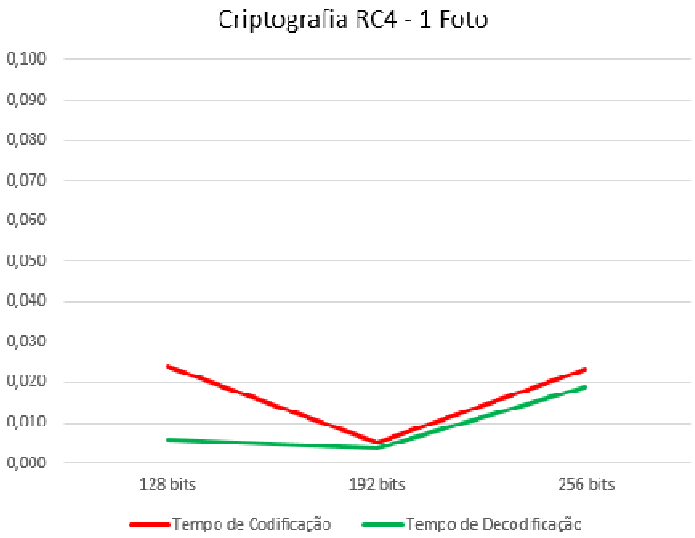

Gráfico 37 - Criptografia de uma imagem de acordo com o tamanho da chave
Fonte: Autoria Própria

Na criptografia de dez imagens a teoria se aplica na função de decriptografia, onde a chave de tamanho 128 ***bits*** apresentou o melhor desempenho, mas deixou de se aplicar nas demais chaves, pois a que apresentou o pior desempenho no tempo decorrido foi a chave de tamanho 192 ***bits***.

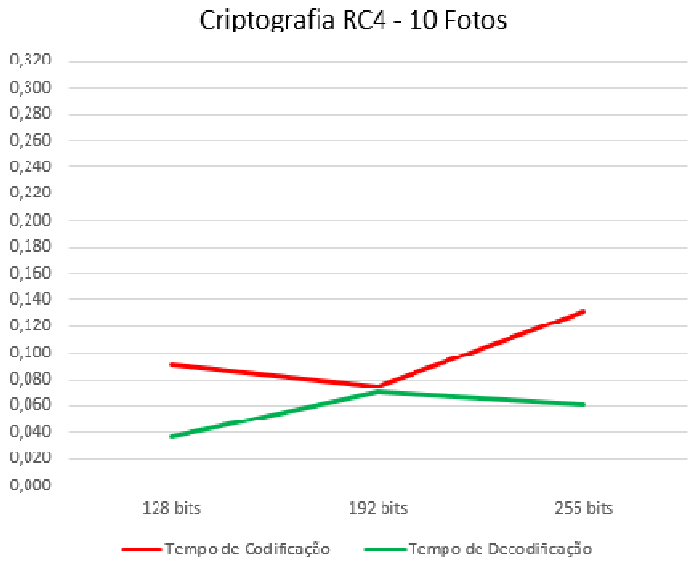

Gráfico 38 - Criptografia de dez imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Na função de criptografia a chave que demonstrou o melhor desempenho de tempo foi a de tamanho 192 ***bits***, seguida da chave de 128 ***bits***. Já a chave de 256 ***bits*** demonstrou o pior desempenho no tempo decorrido nesta função. Os resultados deste teste são apresentados no gráfico 38.

Na utilização do pacote de cinquenta imagens a teoria se aplicou. A chave que mostrou o melhor desempenho no processo foi a de tamanho menor (128 ***bits***) e a chave que mostrou o pior desempenho no processo foi a de tamanho maior (256 ***bits***).

O gráfico abaixo mostra os resultados obtidos neste teste, nota-se que na função de criptografia o crescimento no tempo decorrido foi linear.

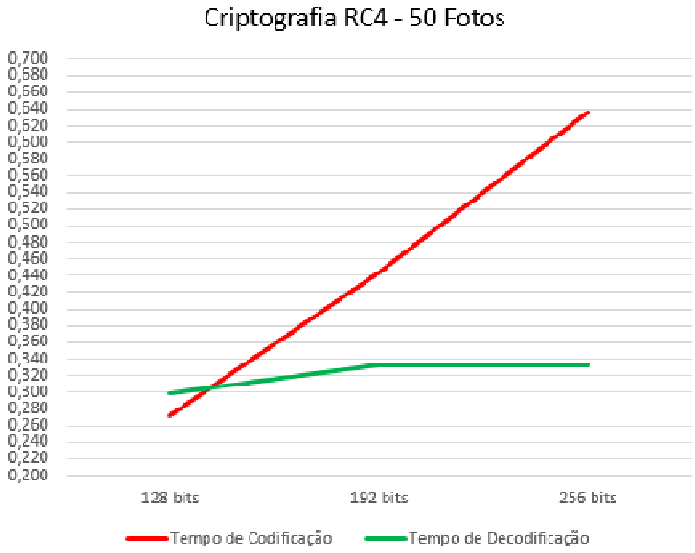

Gráfico 39 - Criptografia de cinquenta imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Na criptografia de cem imagens a teoria se aplicou na função de criptografia, onde a chave de tamanho 128 ***bits*** foi a que apresentou o melhor desempenho, seguida da chave de 192 ***bits***. Na decriptografia a chave que apresentou o melhor desempenho foi a intermediária (192 ***bits***), seguida da chave de 128 ***bits***.

A teoria também se aplicou quando analisada a chave de 256 ***bits***, a qual apresentou o pior desempenho no processo. O gráfico 40 ilustra este resultado.

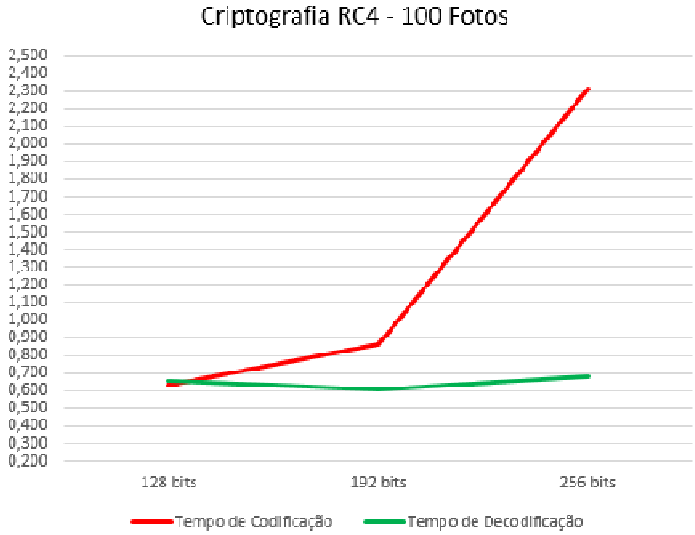

Gráfico 40 - Criptografia de cem imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

Já o gráfico 41 demonstra os resultados obtidos a partir da utilização do pacote de duzentas imagem. Neste caso a teoria se aplicou totalmente. A chave de tamanho menor (128 ***bits***) foi a que apresentou o melhor desempenho no tempo decorrido, seguida da chave de tamanho intermediário (192 ***bits***). Já a chave de tamanho maior (256 ***bits***) foi a que apresentou o pior desempenho no processo.

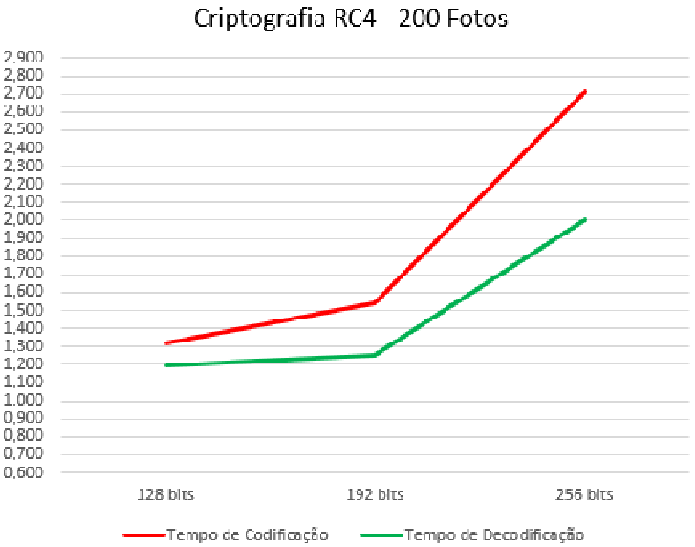

Gráfico 41 - Criptografia de duzentas imagens de acordo com o tamanho da chave
Fonte: Autoria Própria

A seção anterior concluiu que para utilizar a chave de 192 ***bits*** o melhor método é a criptografia imagem-a-imagem e nesta seção os resultados apresentados para esta quantidade de imagens indicam que a melhor chave a ser utilizada é a de tamanho 192 ***bits***. Conclui-se assim, que para a criptografia de imagem-a-imagem com o algoritmo RC4 o melhor tamanho da chave é este.

Concluiu-se também que para os demais tamanhos de chave a melhor vantagem é a criptografia de um pacote de imagens.

Nesta seção os resultados apresentados indicam que o tamanho de chave que apresentou o melhor desempenho para o maior pacote de imagens foi a de tamanho 128 ***bits*** e a chave que apresentou o pior desempenho foi a de 256 ***bits***.

Conclui-se assim que para utilizar um pacote de X imagens a melhor chave a se utilizar no algoritmo RC4 é a de 128 ***bits***.

4.7 ALGORITMO DES

Os testes realizados com o algoritmo DES nesta seção demonstram seu desempenho na geração da imagem criptografada, na geração da imagem decriptografada e no tempo decorrido de acordo com a quantidade de imagens utilizadas.

Tabela 7 - Tamanho da imagem em *bytes* durante o processo com DES

Tamanho da Foto			**Diferença do Tamanho**	
Original	Criptografada	Decriptografada	Criptografada	Decriptografada
52.712	52.720	52.712	8	0
50.102	50.104	50.102	2	0
44.042	44.048	44.042	6	0
49.009	49.016	49.009	7	0
41.090	41.096	41.090	6	0
50.248	50.256	50.248	8	0
40.476	40.480	40.476	4	0
49.858	49.864	49.858	6	0
51.684	51.688	51.684	4	0
27.563	27.568	27.563	5	0

Foi utilizada uma chave única de 64 ***bits*** (a chave possui tamanho de 56 ***bits*** com mais 8 ***bits*** de paridade) e um bloco de mesmo tamanho, conforme é especificado pelo algoritmo. Para demonstrar os resultados obtidos na diferença do tamanho dos arquivos gerados, foi utilizada uma amostra de dez imagens aleatórias, as quais são descritas na tabela acima.

Nota-se que assim como em todos os outros algoritmos, não houve diferença no tamanho da imagem original com o tamanho da imagem decriptografada. As diferenças de tamanho são encontradas entre a imagem original e a imagem criptografada. O pior caso nesta amostragem foi de 8 ***bytes***.

Estas diferenças ocorrem por causa do ***Padding*** inserido na criação da cifra para o processo de criptografia/decriptografia (ver capítulo 2.6.6), ele é o responsável pela inserção de ***bits*** no último bloco, caso o tamanho da informação final não seja um número múltiplo de 64.

Como em todos os outros algoritmos, o DES também se mostrou positivo no método de decriptografia, pois ele desconsiderou todos os ***bits*** de inserção e gerou a imagem decodificada sem alterá-la da imagem original.

O resultado de tempo decorrido de acordo com a quantidade de fotos com o algoritmo DES é apresentado na seção seguinte.

4.7.1 Tempo Decorrido de Acordo com a Quantidade de Imagens

O teste de tempo decorrido realizado neste algoritmo com uma chave de 64 ***bits***, demonstrou que a função de criptografia teve um desempenho no tempo inferior ao da função de decriptografia para uma, dez, cinquenta, cem e duzentas imagens.

No primeiro, a diferença no tempo entre as duas funções foi pequeno. A partir de dez imagens esta diferença é mais significativa, neste caso foi de 0.071s. Com cinquenta imagens foi de 0.121s. Na criptografia de cem imagens a diferença foi de 0.441 e com duzentas imagens foi de 0.180s.

O maior salto registrado foi com a utilização do pacote de cem imagens. O gráfico 42 demonstra a linha do tempo decorrido (eixo vertical) a partir do crescimento da quantidade de imagens (eixo horizontal).

O pior caso registrado neste algoritmo foi a criptografia de duzentas imagens (2.243s), já o melhor caso foi a decriptografia de uma única imagem (0.020s).

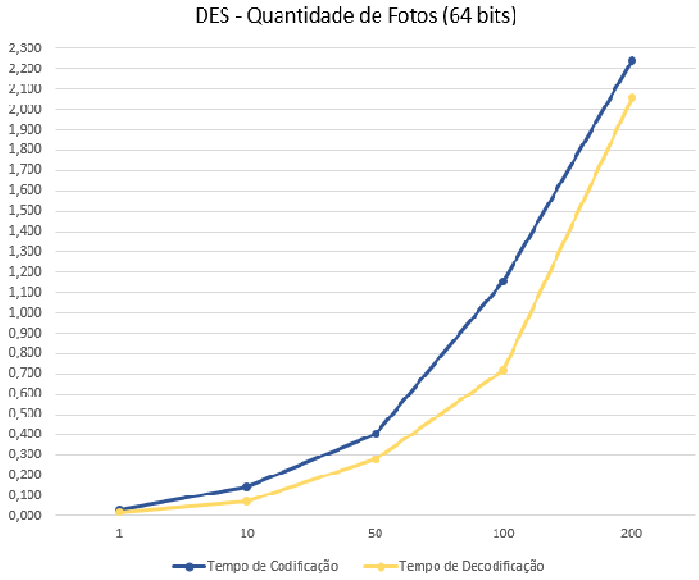

Gráfico 42 - Tempo decorrido com DES e chave de tamanho 64 *bits*
Fonte: Autoria Própria

Para concluir qual é a melhor forma de criptografar por meio do algoritmo DES (imagem-a-imagem ou pacote de imagens) será multiplicado o tempo decorrido para a criptografia de uma única imagem pela quantidade máxima de imagens utilizada neste teste (duzentas).

Como o algoritmo utilizou um tempo de 0.020s para criptografar uma única imagem, conclui-se que a vantagem está na utilização de pacotes de imagens – o pacote com a maior quantidade de imagens levou 2.243s para concluir o processo – pois criptografando imagem-a-imagem o algoritmo levaria aproximadamente 4.000s para realizar a função.

Apesar disto a utilização deste algoritmo é inviável, pois possui um tamanho de chave muito pequeno e fixo, o qual já foi quebrado por “força-bruta”, conforme foi descrito no capítulo 2.4.1.

4.8 ALGORITMO 3DES

Nos testes realizados com o algoritmo 3DES foi utilizada a chave de 192 bits – são três chaves de 64 bits (tamanho da chave do DES – no qual este baseia-se). Foram analisados: o tempo decorrido de acordo com a quantidade de imagens e a diferença no tamanho das imagens criptografadas e decriptografadas.

Para demonstrar os resultados, foram escolhidas dez imagens aleatórias. Como o ocorrido nos demais algoritmos, o 3DES apresentou um bom desempenho com relação ao tamanho da imagem decriptografada, isto pode ser observado na tabela abaixo – não houve diferença no tamanho da imagem original com a imagem decodificada.

Tabela 8 - Tamanho da imagem em *bytes* durante o processo com 3DES

Tamanho da Foto			**Diferença do Tamanho**	
Original	Criptografada	Decriptografada	Criptografada	Decriptografada
51.521	51.528	51.521	7	0
42.190	42.192	42.190	2	0
47.828	47.832	47.828	4	0
45.282	45.288	45.282	6	0
51.302	51.304	51.302	2	0
46.580	46.584	46.580	4	0
45.382	45.384	45.382	2	0
47.257	47.264	47.257	7	0
44.106	44.112	44.106	6	0
46.749	46.752	46.749	3	0

A diferença ocorreu com a imagem criptografada. O algoritmo apresentou como pior resultado um acréscimo de 7 ***bytes***. Esta diferença ocorre porque, se o tamanho da informação final (***bits*** da imagem original "misturados" com os ***bits*** da chave) não for um número múltiplo de 64 (tamanho do bloco) o último bloco deve ser completado com ***bits*** adicionais.

Mesmo assim o algoritmo demonstrou um bom resultado na função de decriptografia, pois ela desconsiderou todos os ***bits*** adicionais e gerou a imagem decodificada exatamente igual a original.

O teste de tempo decorrido de acordo com a quantidade de imagens é apresentado no capítulo seguinte.

4.8.1 Tempo Decorrido de Acordo com a Quantidade de Imagens

O teste realizado com o algoritmo 3DES com uma chave de 192 ***bits*** demonstrou que a função de decriptografia teve um desempenho superior de tempo comparada a criptografia. Utilizando uma imagem a diferença no tempo das duas funções foi de 0.057s.

Já no pacote de dez imagens a diferença foi de 0.087s. Com cinquenta imagens a função de decriptografia foi 0.150s superior a criptografia. Com cem imagens a diferença foi de 0.202s e com duzentas imagens foi de 0.180s.

O melhor caso deste algoritmo foi na decriptografia de uma única imagem (0.030s) e o pior caso ocorreu na criptografia do pacote de duzentas imagens (2.678s). O gráfico abaixo demonstra a curva do tempo decorrido (eixo vertical) de acordo com a quantidade de imagens (eixo horizontal).

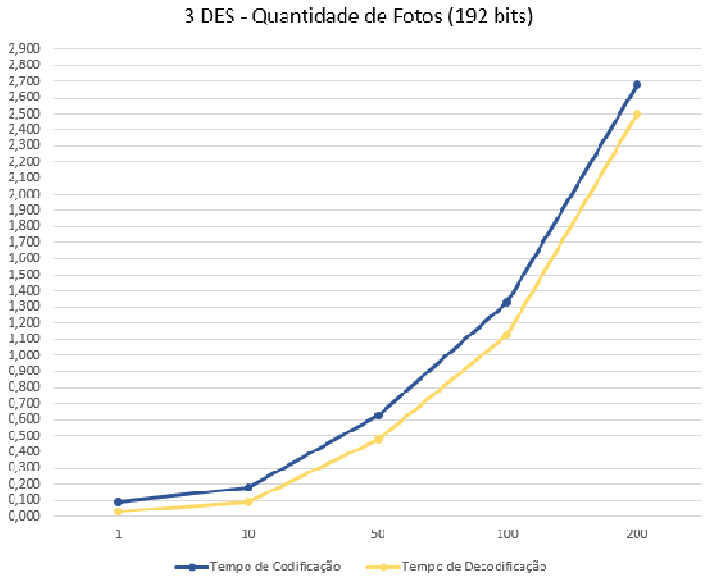

Gráfico 43 - Tempo decorrido com 3DES e chave de tamanho 192 *bits*
Fonte: Autoria Própria

Para concluir esta seção é necessário verificar se o algoritmo apresentaria melhor desempenho de tempo para uma quantidade X de imagens se criptografasse imagem-a-imagem ou um pacote de imagens.

Para isso é multiplicado o tempo decorrido para a criptografia de uma única imagem pela quantidade máxima de imagens testadas nesta seção (duzentas). Então, no caso do algoritmo 3DES conclui-se que a melhor forma de criptografar uma quantidade X de imagens é por meio de pacotes.

Se fossem criptografadas duzentas imagens uma-a-uma o algoritmo levaria aproximadamente 6.000s e ele apresentou o desempenho de tempo de 2.678s para criptografar o pacote de duzentas imagens.

4.9 ALGORITMO RSA

Os testes realizados com a implementação do algoritmo RSA não apresentaram resultados positivos. Foram feitas três tentativas de encriptação de uma única imagem, utilizando chaves de tamanho 2048, 4096 e 16384 ***bits***. Tendo em vista a forma como foi realizada a leitura e a gravação dos ***bits*** das imagens descrita no capítulo 2.7, o algoritmo retornou os erros elencados abaixo.

Para a chave de tamanho 2048 ***bits*** foi realizada a criação do par de chaves e da cifra, mas no momento da encriptação não foi possível realizar a operação porque o bloco de informação não pode possuir um tamanho superior a 245 ***bytes***, e no caso da imagem o bloco possuí 50.350 ***bytes***. Na figura abaixo é demonstrado o erro que a classe retornou.

```
run:
Iniciando Codificação...
Exception in thread "main" javax.crypto.IllegalBlockSizeException: Data must not be longer than 245 bytes
        at com.sun.crypto.provider.RSACipher.doFinal(RSACipher.java:344)
        at com.sun.crypto.provider.RSACipher.engineDoFinal(RSACipher.java:389)
        at javax.crypto.Cipher.doFinal(Cipher.java:2048)
        at Assimetrico.EncriptaDecriptaRSA.encrypt(EncriptaDecriptaRSA.java:51)
        at Assimetrico.EncriptaDecriptaRSA.main(EncriptaDecriptaRSA.java:89)
Java Result: 1
CONSTRUÍDO COM SUCESSO (tempo total: 2 segundos)
```

Figura 8 - Criptografia utilizando o algoritmo RSA e uma chave de 2048 bits
Fonte: Autoria Própria

O mesmo erro também ocorre na utilização da chave de 4096 ***bits*** para a tentativa da encriptação da mesma imagem, onde o algoritmo retorna que o bloco de

informação a ser encriptada não pode possuir mais do que 501 ***bytes*** de tamanho, conforme pode ser visto na figura 9.

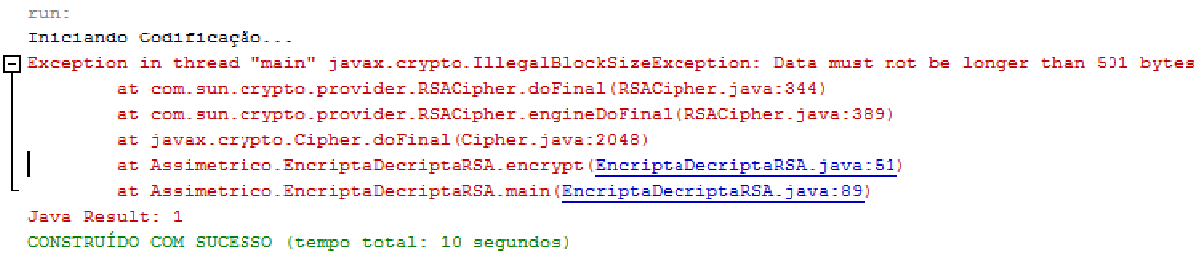

Figura 9 - Criptografia utilizando o algoritmo RSA e uma chave de 4096 bits
Fonte: Autoria Própria

Na terceira tentativa foi utilizada a maior chave suportada pelo algoritmo, de tamanho 16834 ***bits*** para a encriptação da mesma imagem de 50.350 ***bytes***, mas ainda assim o mesmo erro é apresentado, para este caso o bloco de informação a ser criptografado pode ter até 2.037 ***bytes*** de tamanho. A figura abaixo demonstra o erro apresentado pela classe.

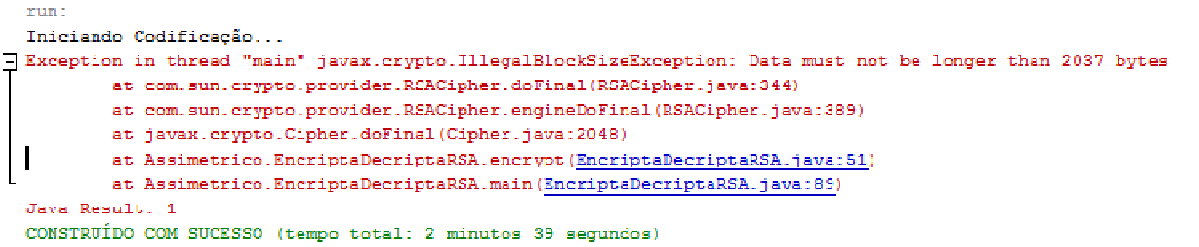

Figura 10 - Criptografia utilizando o algoritmo RSA e uma chave de 16384 bits
Fonte: Autoria Própria

O tamanho do bloco no caso do RSA é limitado ao tamanho da chave. Há uma forma de criptografar imagens utilizando este algoritmo implementada por Silva et al. a qual é descrita no artigo “Criptografia assimétrica de imagens utilizando algoritmo RSA”. Esta técnica consiste na cifragem de cada ***pixel*** da imagem e a partir deles a geração de um arquivo de imagem.

Esta técnica não se aplica ao escopo deste trabalho, pois para este foram utilizadas classes próprias do Java para leitura e gravação dos ***bits*** da imagem. Além

disso os resultados apresentados pelos autores demonstram que houve um aumento considerável no tamanho da imagem criptografada, o que não acrescentaria ao desenvolvimento deste trabalho.

Geralmente o algoritmo RSA é utilizado junto a um dos algoritmos simétricos, onde: é gerada uma chave privada com o algoritmo simétrico; a informação é encriptada por meio desta chave; é gerado o par de chaves com o RSA; a chave gerada com o algoritmo simétrico é encriptada por meio da chave pública.

São encaminhadas a chave e a informação codificadas pelo canal inseguro de comunicação; o destinatário decripta a chave com o algoritmo RSA e com a chave decriptada é possível obter os dados originais da informação.

> "Algoritmos assimétricos vieram resolver o problema de compartilhamento de chaves que existia com os algoritmos simétricos. No mundo real é interessante usar as duas técnicas em conjunto tirando o que há de melhor em ambas. Embora os algoritmos assimétricos sejam muito mais lentos que os simétricos, sua utilização em conjunto é amplamente difundida (BRAZIL, 2007, p. 24)."

A criptografia com este algoritmo é mais comumente utilizada em assinatura e certificados digitais, e em criptografia de pequenas quantidades de ***bits***.

Sendo assim, conclui-se que no escopo deste trabalho, a utilização deste algoritmo só será viável se combinado ao algoritmo simétrico que apresentou os melhores resultados – conforme pode ser visto no capítulo 6.2 – para garantir a segurança da transmissão da chave na comunicação e a rapidez no processo de criptografia.

5 COMPARAÇÃO DOS RESULTADOS

Neste capítulo é realizado o comparativo do desempenho de todos os algoritmos testados neste trabalho, de acordo com o tamanho de chave utilizado (128, 192 e 256 ***bits***). No caso do DES, que utiliza uma chave de 64 ***bits***, foi feita uma comparação desta com o menor tamanho de chave dos demais algoritmos (128 ***bits***).

Ao final dela será possível precisar qual algoritmo é mais vantajoso se utilizado com determinado tamanho de chave. Abaixo são descritos os resultados obtidos nas comparações.

5.1 CHAVE DE 128 BITS

Na comparação dos algoritmos que criptografam com uma chave de 128 ***bits*** (AES, Blowfish, RC2, RC5 e RC4) e 64 bits (DES), uma única imagem, o processo como um todo não pode ser analisado – cada função apresenta uma particularidade.

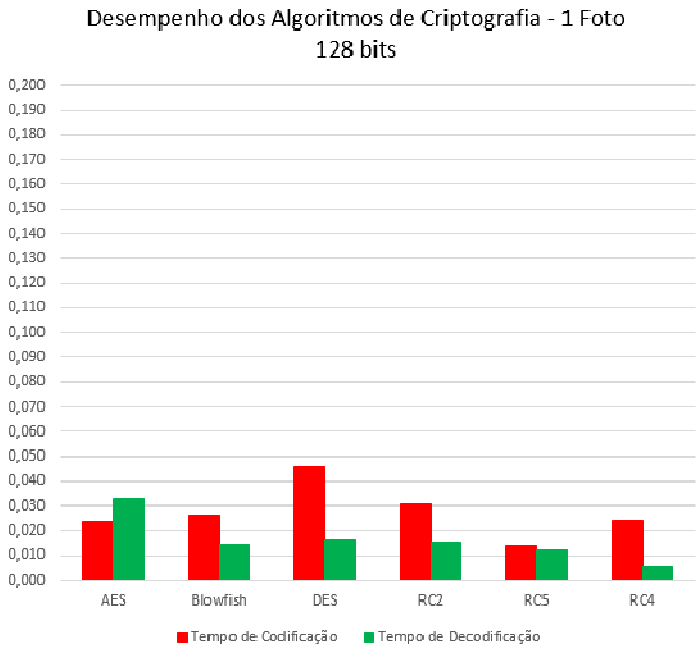

Gráfico 44 - Algoritmos de criptografia para uma imagem com chave de 128
Fonte: Autoria Própria

O algoritmo que apresentou o melhor desempenho na função de criptografia foi o RC5, seguido pelo AES, RC4, Blowfish e RC2. O algoritmo que apresentou o pior desempenho nesta função foi o DES.

Na decriptografia o RC4 foi o algoritmo que apresentou o melhor desempenho seguido pelo RC5, Blowfish, RC2 e DES. O AES foi o algoritmo que apresentou o pior desempenho nesta função. O gráfico 44 ilustra estes resultados.

O gráfico 45 demonstra os resultados obtidos nos testes com dez imagens. Neste caso os algoritmos também apresentaram desempenhos diferentes entre as duas funções. Na criptografia o RC5 foi o algoritmo que apresentou o melhor resultado, seguido pelo RC2, DES, RC4 e Blowfish, já o AES foi o que demonstrou o pior desempenho na função.

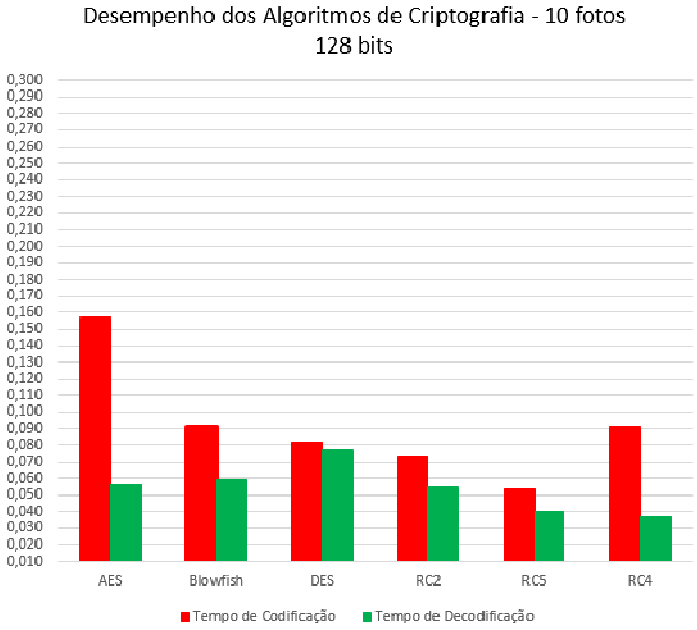

Gráfico 45 - Algoritmos de criptografia para dez imagens com chave de 128
Fonte: Autoria Própria

Na decriptografia o algoritmo que apresentou o melhor resultado foi o RC4, seguido pelo RC5, RC2, AES e Blowfish, já o DES foi o algoritmo que apresentou o pior desempenho.

Na função de criptografia de cinquenta imagens o algoritmo que apresentou o melhor desempenho foi o RC4, seguido pelo RC5, AES, RC2 e Blowfish, já o que demonstrou o pior resultado foi o DES.

Na função de decriptografia o algoritmo que apresentou o melhor desempenho foi o RC5, seguido pelo DES, RC4, AES e RC2, já o algoritmo que apresentou o pior desempenho foi o Blowfish. O gráfico 46 demonstra estes resultados.

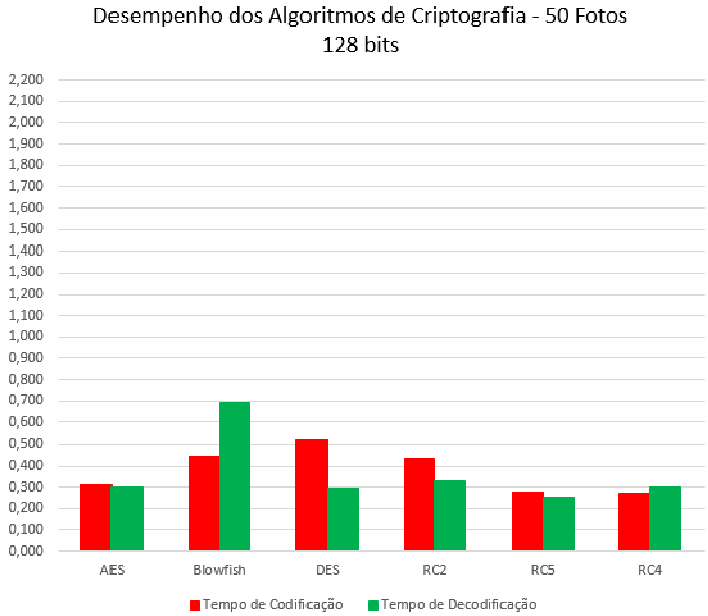

Gráfico 46 - Algoritmos de criptografia para cinquenta imagens com chave de 128
Fonte: Autoria Própria

Utilizando um pacote de cem fotos, o algoritmo que apresentou o melhor desempenho em todo o processo foi o RC5. Os segundos melhores foram o AES e o RC4, onde o primeiro se mostrou melhor na função de decriptografia e o segundo na função de criptografia.

O terceiro melhor foi o algoritmo DES. Já o algoritmo que demonstrou o pior resultado na função de criptografia foi o RC2 e na função de decriptografia foi o algoritmo Blowfish. Estes resultados podem ser analisados no gráfico 47.

Na função de criptografia de duzentas imagens o algoritmo que demonstrou o melhor resultado foi o RC5, seguido pelo AES, RC4, Blowfish e DES, já o que decorreu o maior tempo nesta função foi o algoritmo RC2.

Na função de decriptografia o algoritmo que ocupou o menor tempo foi o RC4, seguido pelo AES, Blowfish, RC2 e RC5, já o algoritmo que demonstrou o pior resultado foi o DES. O gráfico 48 demonstra estes resultados.

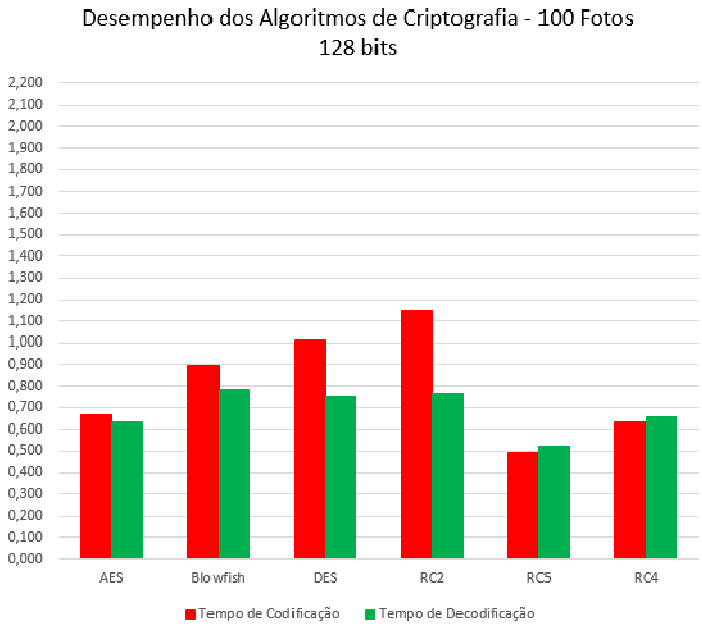

Gráfico 47 - Algoritmos de criptografia para cem imagens com chave de 128
Fonte: Autoria Própria

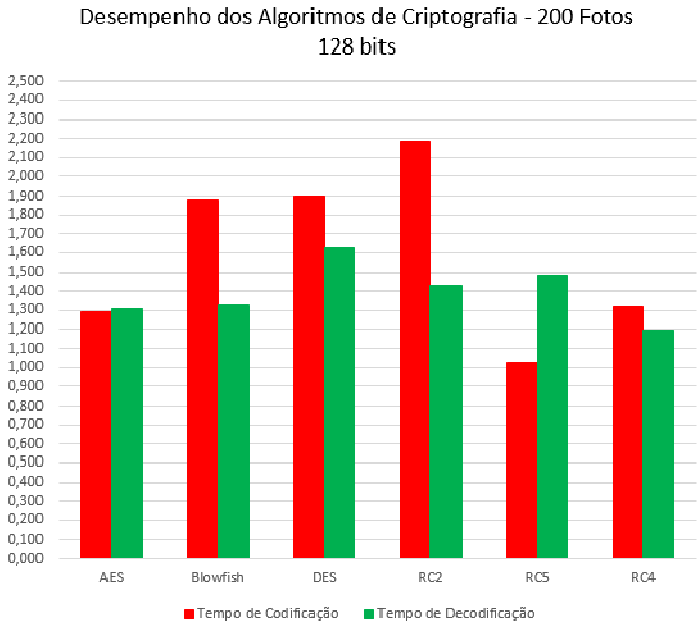

Gráfico 48 - Algoritmos de criptografia para duzentas imagens com chave de 128
Fonte: Autoria Própria

Na utilização da chave de tamanho 128 ***bits*** os resultados apresentados pelos algoritmos foram bem variados. O RC5 foi o algoritmo que apresentou o melhor resultado em seis dos dez testes realizados e o RC4 se apresentou melhor em quatro dos testes.

Nos dez testes realizados o DES apresentou o pior resultado em quatro deles, o AES em dois deles, o Blowfish também em dois deles e o RC2 também demonstrou o pior resultado em dois dos testes.

Apesar do DES utilizar uma chave com metade do tamanho dos demais, em nenhum dos casos ele apresentou um desempenho de tempo melhor, ao contrário, foi o algoritmo que demonstrou um resultado tempo inferior em mais testes (quatro) se comparado aos demais.

Conclui-se assim que é melhor utilizar o algoritmo de criptografia RC5 com uma chave de 128 bits. Já o pior algoritmo a se utilizar com uma chave deste tamanho não pode ser precisado, pois nos testes realizados os algoritmos AES, Blowfish e RC2 apresentaram desempenhos insatisfatórios em dois testes cada um.

5.2 CHAVE DE 192 BITS

Comparando todos os algoritmos que trabalham com uma chave de 192 ***bits*** (AES, Blowfish, 3DES, RC2, RC4 e RC5), para a criptografia de uma única imagem os resultados foram: o algoritmo 3DES foi o que apresentou o pior desempenho no processo como um todo.

Já os algoritmos RC4 e RC5 tiveram um desempenho igual no processo, sendo estes os melhores. O algoritmo RC2 teve um desempenho na criptografia igual aos outros RC's descritos, já na decriptografia ele foi 0.001s mais lento do que os outros.

O AES teve um desempenho igual aos RC's na criptografia e um resultado igual ao 3DES na decriptografia, ele mostrou o melhor e o pior desempenho no mesmo processo, sendo este o único algoritmo que a função de decriptografia ocupou tempo superior a criptografia.

Já o Blowfish foi intermediário, não apresentando nem o melhor nem o pior desempenho em nenhuma das funções no processo. O gráfico 49 apresenta os resultados obtidos.

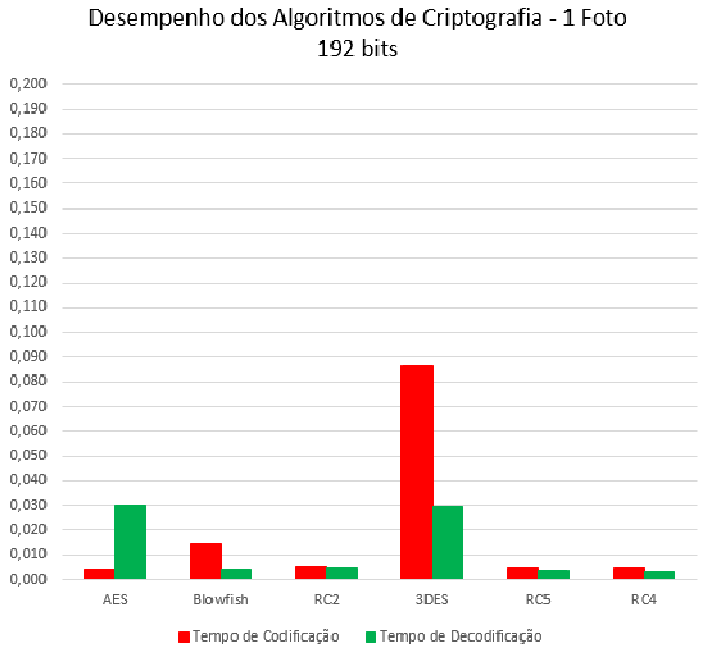

Gráfico 49 - Algoritmos de criptografia para uma imagem com chave de 192
Fonte: Autoria Própria

Na criptografia de dez imagens o algoritmo que apresentou o melhor desempenho no processo foi o RC5, seguido pelo AES – que neste caso a função de criptografia ocupou tempo superior a decriptografia.

O terceiro com melhor desempenho foi o RC4, que apesar de trabalhar com fluxo ao invés de blocos, para esta quantidade de imagens e este tamanho de chave não mostrou o melhor desempenho.

O Blowfish apresentou um desempenho igual ao RC4 na função de criptografia, mas na decriptografia ele ocupou 0.003s a mais que o algoritmo anterior. O algoritmo RC2 foi o penúltimo colocado e o algoritmo 3DES foi o que apresentou o pior desempenho neste conjunto de testes. O gráfico 50 demonstra estes resultados.

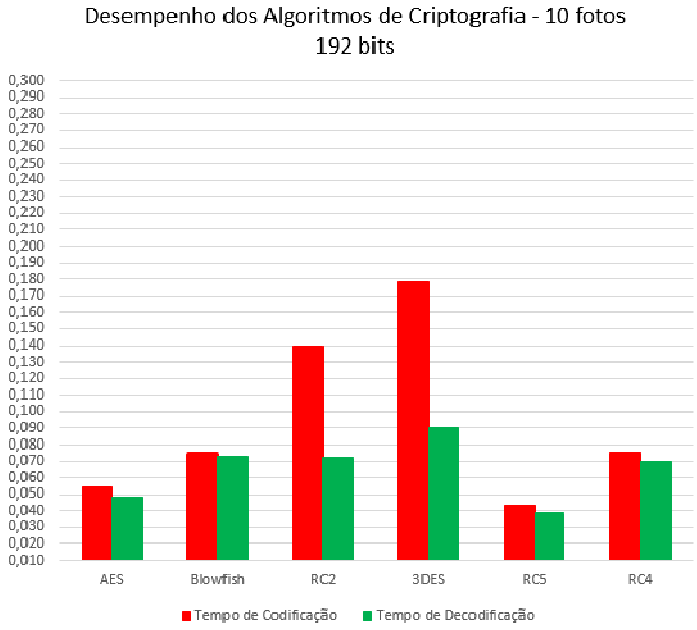

Gráfico 50 - Algoritmos de criptografia para dez imagens com chave de 192
Fonte: Autoria Própria

Na criptografia do pacote de cinquenta imagens o algoritmo RC5 continua a demonstrar o melhor desempenho no processo. O RC2 e o RC4 demonstraram os segundos melhores resultados em uma das funções, o primeiro se apresentou melhor na função de criptografia e o segundo foi melhor na decriptografia.

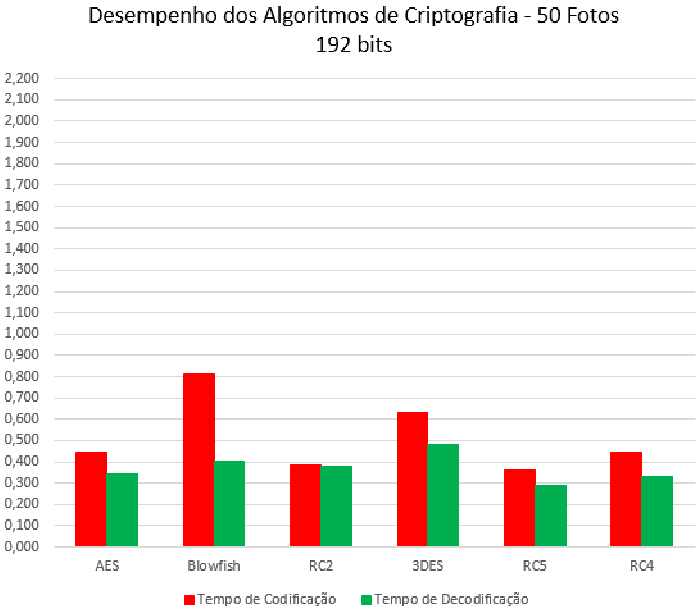

Gráfico 51 - Algoritmos de criptografia para cinquenta imagens com chave de 192
Fonte: Autoria Própria

O AES apresentou o quarto melhor resultado, mas na função de decriptografia ele se demonstrou melhor que o algoritmo RC2. O Blowfish teve um desempenho inferior ao do algoritmo 3DES na função de criptografia, já na decriptografia o 3DES foi o que apresentou o pior desempenho.

Estes foram os dois algoritmos que apresentaram o pior tempo neste teste. Os resultados são apresentados no gráfico 51.

Na criptografia de cem imagens o processo como um todo não pode ser verificado, cada algoritmo mostrou um resultado superior aos outros em uma das duas funções. Na função de criptografia o algoritmo que se mostrou melhor foi o AES, seguido do Blowfish, RC2, RC4, RC5, nesta sequência respectivamente.

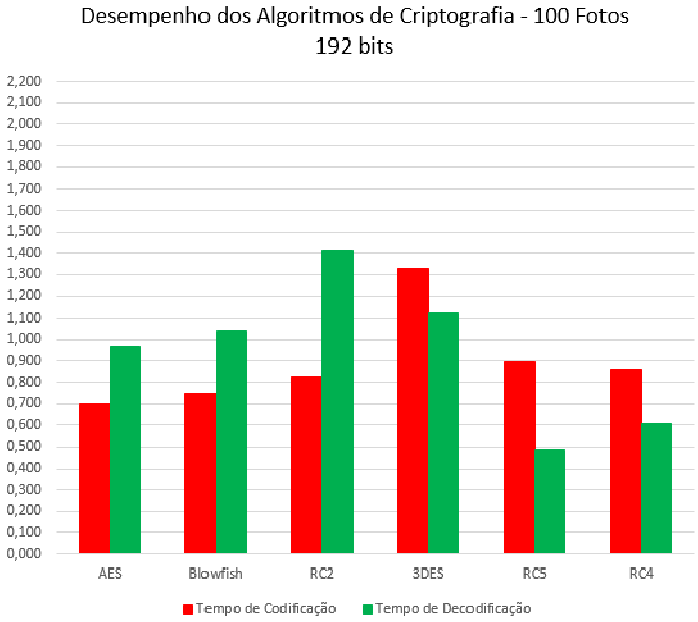

Gráfico 52 - Algoritmos de criptografia para cem imagens com chave de 192
Fonte: Autoria Própria

O 3DES foi o que demonstrou o pior resultado nesta função. Já na decriptografia o algoritmo que apresentou o melhor desempenho foi o RC5 seguido do RC4, AES, Blowfish e 3DES.

Neste teste o algoritmo RC2 foi o que apresentou o pior desempenho na função de decriptografia. Os resultados são apresentados no gráfico 52.

O gráfico 53 demonstra os resultados obtidos na criptografia de duzentas imagens, neste caso o algoritmo RC5 continua a demonstrar o melhor resultado no processo. O segundo colocado foi o RC4, seguido pelo Blowfish e pelo AES. O Blowfish apresentou melhor desempenho na função de criptografia se comparado com o AES.

Já na função de decriptografia foi o AES que demonstrou o melhor desempenho sobre o Blowfish. O penúltimo colocado neste teste foi o algoritmo RC2. O 3DES foi o que apresentou o pior desempenho no processo.

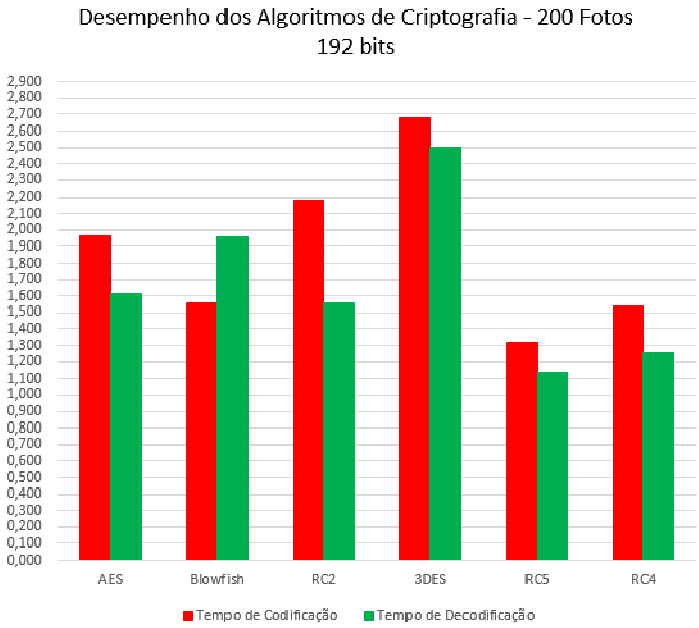

Gráfico 53 - Algoritmos de criptografia para duzentas imagens com chave de 192
Fonte: Autoria Própria

Nota-se que na utilização de uma chave de tamanho 192 ***bits*** os algoritmos RC5 e AES demonstraram os melhores desempenhos – o RC4 também demonstrou o melhor desempenho na função de decriptografia de uma imagem.

Todos os outros algoritmos se apresentaram intermediários. Já o algoritmo que mostrou o pior desempenho foi o 3DES – os algoritmos AES, Blowfish e RC2 também demonstraram o pior tempo decorrido em alguns casos.

Na criptografia de imagens utilizando uma chave de 192 ***bits*** os algoritmos apresentam resultados variáveis em todos os casos. Alguns se demonstram melhores em uma função e piores em outra.

Pode-se concluir então que, utilizando uma chave de tamanho 192 ***bits*** é melhor utilizar o algoritmo RC5, pois ele apresentou o melhor desempenho em oito dos dez testes realizados. Já o algoritmo que não deve ser utilizado neste tipo de criptografia é o 3DES, pois ele apresentou o pior desempenho em oito dos dez testes realizados.

5.3 CHAVE DE 256 BITS

Utilizando uma única imagem para o teste dos algoritmos que criptografam com uma chave de 256 ***bits*** (AES, Blowfish, RC2, RC5 e RC4), para cada função um determinado algoritmo se sobressaiu aos outros.

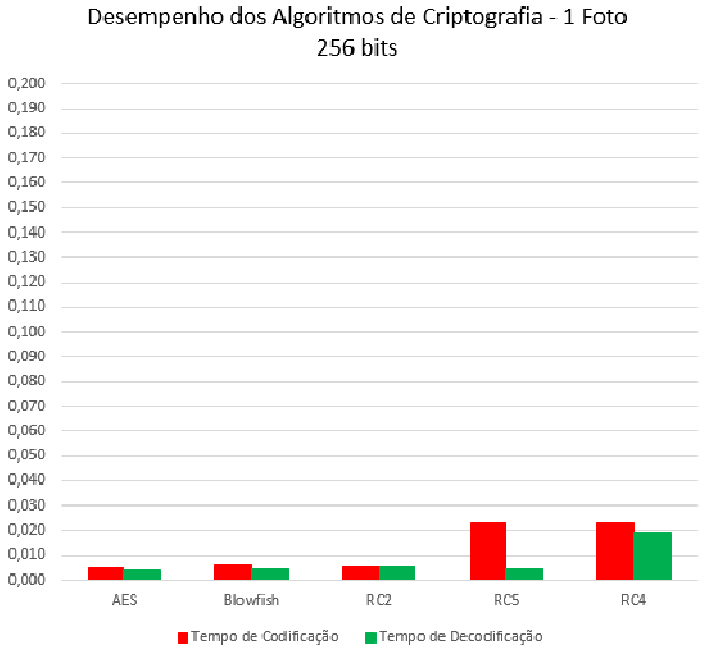

Gráfico 54 - Algoritmos de criptografia para uma imagem com chave de 256
Fonte: Autoria Própria

Na função de criptografia o que teve o melhor desempenho foi o AES, seguido do Blowfish e do RC2, os algoritmos RC5 e o RC4 apresentaram o pior desempenho nesta função.

Já na função de decriptografia os algoritmos que apresentaram o melhor desempenho foram o AES e o RC5, seguidos pelo Blowfish e pelo RC2, o algoritmo RC4 foi o que apresentou o pior desempenho nesta função. Os resultados são apresentados no gráfico 54.

O gráfico 55 demonstra os resultados apresentados na criptografia de dez imagens. Neste teste também não é possível analisar o processo como um todo, pois cada algoritmo se sobressaiu de uma forma diferente em cada função.

Para a criptografia o algoritmo que demonstrou o melhor desempenho foi o AES, seguido pelo RC5, Blowfish e RC2. Logo, o que demonstrou o pior desempenho foi o RC4.

Na função de decriptografia o algoritmo que apresentou o melhor desempenho foi o RC5, seguido pelo RC4, AES e RC2, e o algoritmo que ocupou o maior intervalo de tempo para realizar a função foi o Blowfish.

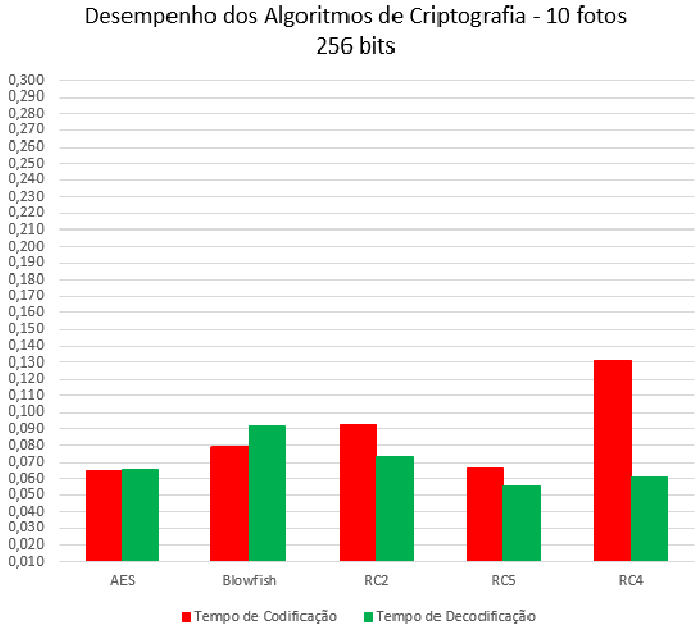

Gráfico 55 - Algoritmos de criptografia para dez imagens com chave de 256
Fonte: Autoria Própria

Utilizando um pacote de cinquenta imagens, também não é possível verificar qual algoritmo se sobressaiu no processo, pois cada um teve suas particularidades em

cada função. O algoritmo RC5 foi o que apresentou o melhor desempenho na função de criptografia, seguido pelo AES, RC2 e Blowfish.

O que apresentou o pior desempenho foi o algoritmo RC4. Já na função de decriptografia o algoritmo RC4 foi o que se sobressaiu, seguido pelo RC2, RC5 e Blowfish, o algoritmo AES apresentou o pior desempenho nesta função.

Neste caso o RC4 apresentou o melhor e o pior desempenho no processo. O gráfico 56 demonstra estes resultados.

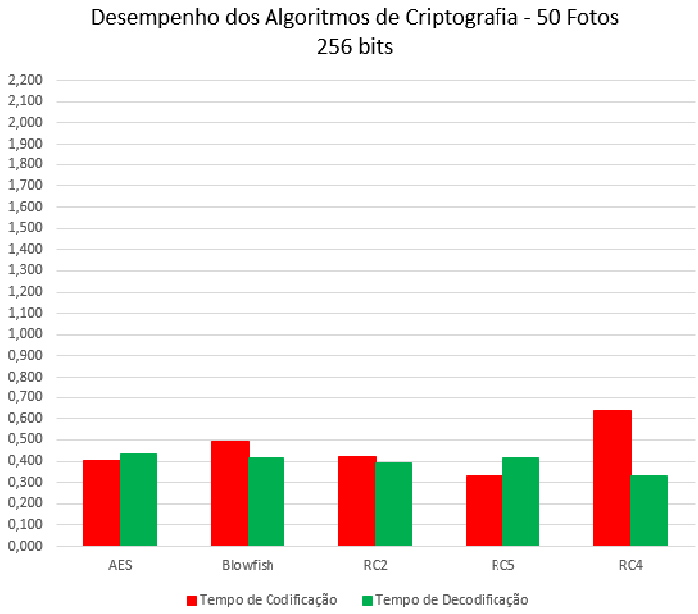

Gráfico 56 - Algoritmos de criptografia para cinquenta imagens com chave de 256
Fonte: Autoria Própria

No caso da criptografia de cem imagens, também não é possível analisar o processo como um todo, cada algoritmo teve suas particularidades em cada função. O AES foi o que apresentou o melhor desempenho na função de criptografia, seguido pelo Blowfish, RC5 e RC2, sendo o RC4 o algoritmo que demonstrou o pior desempenho na função.

Já na decriptografia o algoritmo que se sobressaiu foi o RC5, seguido pelo RC4, RC2 e Blowfish, o algoritmo AES foi o que apresentou o pior desempenho nesta função. Neste caso o AES apresentou o melhor e o pior desempenho no processo. O gráfico 57 ilustra estes resultados.

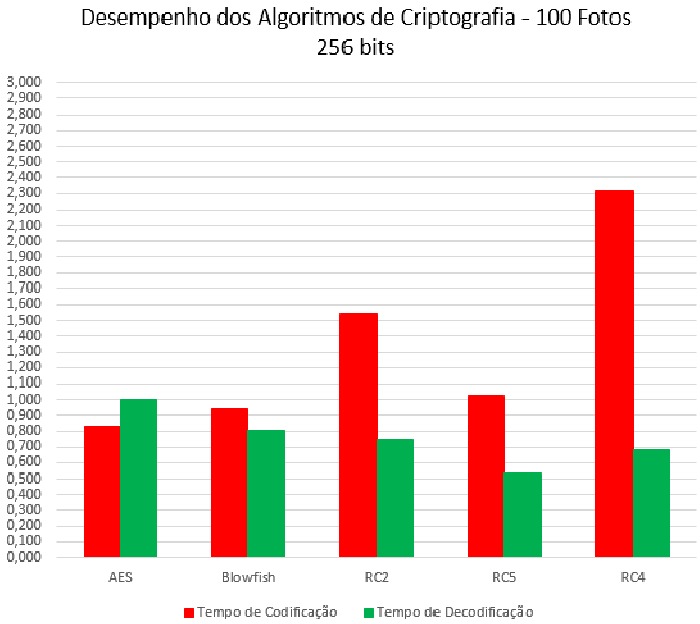

Gráfico 57 - Algoritmos de criptografia para cem imagens com chave de 256
Fonte: Autoria Própria

O gráfico 58 demonstra os resultados obtidos criptografando um pacote de duzentas imagens. O algoritmo que apresentou o melhor desempenho foi o RC5.

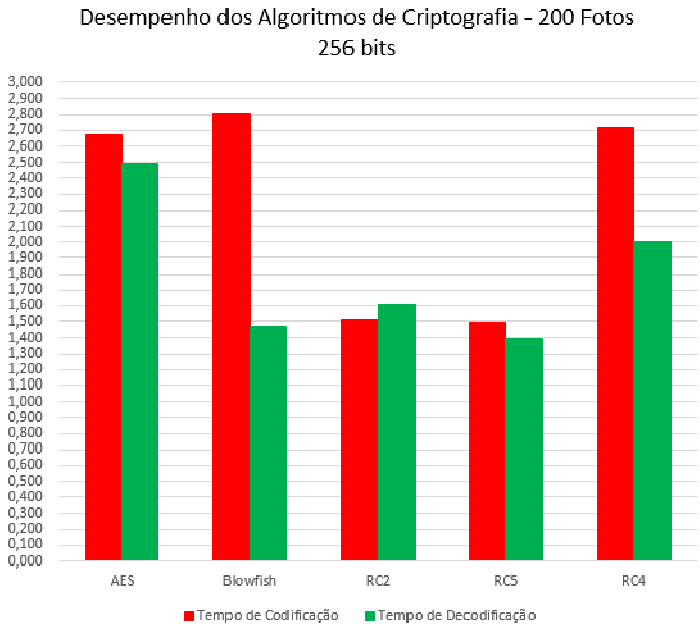

Gráfico 58 - Algoritmos de criptografia para duzentas imagens com chave de 256
Fonte: Autoria Própria

Os segundos colocados foram o Blowfish e o RC2. O primeiro demonstrou o melhor desempenho na função de decriptografia e o segundo na função de criptografia.

Já os algoritmos que apresentaram o pior desempenho foram o AES e o RC4. O primeiro teve seu pior desempenho na função de decriptografia e o segundo na função de criptografia.

Nos testes realizados com a chave de 256 ***bits*** é possível observar que em quase todos os testes não foi possível fazer uma verificação do processo como um todo, pois cada algoritmo se comportou de uma forma diferente em cada função.

Fazendo uma comparação de todos os resultados o algoritmo que apresentou o melhor desempenho foi o RC5 – os algoritmos AES e RC4 também apresentaram bons desempenhos em alguns casos.

Já o algoritmo que apresentou o pior desempenho nos testes foi o RC4 – os algoritmos Blowfish, AES e RC2 também apresentaram desempenhos ruins em alguns casos. Cada algoritmo teve sua particularidade utilizando este tamanho de chave.

A variação de desempenho dada a quantidade de imagens foi bem significativa. De acordo com os resultados apresentados, conclui-se que para utilizar a chave que apresenta a melhor segurança, é mais viável criptografar por meio do algoritmo RC5, pois, dos dez testes realizados ele apresentou o melhor desempenho em seis deles.

Já o algoritmo que não tem sua utilização viável com este tamanho de chaves é o RC4, pois apresentou o pior desempenho em quatro dos dez testes realizados (o AES foi pior em três dos testes, o Blowfish em dois e o RC2 em um).

6 CONCLUSÃO

Na criptografia de grandes quantidades de ***bits*** os resultados obtidos são variados, cada algoritmo apresentou uma particularidade especifica em cada caso.

Este capítulo expõe as considerações finais do trabalho, as dificuldades encontradas para elaborá-lo, quais os resultados obtidos por meio dele e quais são os possíveis trabalhos a serem realizados futuramente, tomando como base os resultados obtidos.

6.1 DIFICULDADES ENCONTRADAS

As dificuldades para a realização deste trabalho foram:

- Encontrar trabalhos recentes sobre criptografia com chaves simétricas, a maioria dos trabalhos (artigos, trabalhos de conclusão de curso, monografias) são antigos.
- Encontrar trabalhos técnicos científicos que abordassem o tema especifico da criptografia de grandes quantidades de ***bits***, principalmente imagens.
- Os trabalhos encontrados (que abordavam a criptografia de imagens) demonstravam somente a utilização de algoritmos assimétricos. Para os simétricos não foi encontrado nenhum artigo, trabalho de conclusão de curso ou monografia que abordasse o assunto específico da criptografia de imagens.
- Encontrar formas para a criptografia de imagens. Foram encontradas apenas duas – uma implementada neste trabalho e outra que criptografava as bandas de RGB da imagem (a qual não foi possível implementar, pois haviam poucas informações).
- Na implementação dos algoritmos, houve a dificuldade em encontrar a forma de definir o tamanho da chave (para poder utilizar os três tamanhos definidos no escopo).
- Os resultados dos testes de tamanho do arquivo em cada um dos algoritmos simétricos retornavam exatamente iguais para os três tamanhos de chave.

Não foram encontrados trabalhos que abordassem esse quesito, mas foi concluído que este era um desempenho normal visto que eles utilizam tamanho de bloco fixo.

- Encontrar a forma de funcionamento do algoritmo RC2, pois pouco é detalhado sobre ele nos trabalhos pesquisados.
- Houve a dúvida se poderiam ser utilizados os algoritmos RC's nos testes do trabalho, pois as informações encontradas eram de que eles são softwares proprietários. Como a implementação dos algoritmos foi feita pela API de criptografia do Java, então os mesmos podem ser testados, tendo em vista que a JCA é pública.
- Encontrar dados detalhados sobre as classes da JCA do Java e como elas funcionam.

6.2 CONSIDERAÇÕES FINAIS

De acordo com o escopo proposto neste trabalho, foram realizados testes utilizando três tamanhos de chave: 128, 192 e 256 ***bits*** em cinco algoritmos simétricos: AES, Blowfish, RC2, RC4 e RC5. Também foram realizados testes com os algoritmos simétricos que possuem chave de tamanho fixo: DES com chave de 64 ***bits*** e 3DES com chave de 192 ***bits***.

O único algoritmo assimétrico testado neste trabalho foi o RSA com três tamanhos de chaves: 2048, 4096 e 16384 ***bits***. Os tamanhos dos blocos utilizados para os algoritmos (menos o RC4) seguiram o padrão da implementação da classe ***Cipher*** do Java.

Nos testes realizados com os algoritmos simétricos com chave de 128 ***bits***, os algoritmos que demonstraram o melhor desempenho foram o RC5 e o RC4. Conclui-se, de acordo com os testes, que o melhor caso para este tamanho de chave se encontra na criptografia de um pacote de imagens.

Os algoritmos que demonstraram desempenho insatisfatório para este tamanho de chave foram o DES, o AES, o Blowfish e o RC2.

Para a chave de 192 ***bits*** o RC5 também demonstrou o melhor desempenho em comparação aos outros testados na maioria dos casos (em um dos casos o RC4

teve o mesmo desempenho), nos demais casos o AES que foi o melhor. Neste caso, os algoritmos RC4 e RC5 apresentam seu melhor caso na criptografia de imagem-a-imagem e o algoritmo AES na criptografia de um pacote de imagens.

Já os algoritmos que demonstraram os piores desempenhos nos testes com este tamanho de chave foram o 3DES (em um dos casos o AES teve o mesmo desempenho), o RC2 e o Blowfish.

Para a chave de 256 ***bits*** os algoritmos que demonstraram os melhores desempenhos foram o RC5 (em um dos casos o RC4 teve o mesmo desempenho) e o AES. Novamente, para este tamanho de chave, conclui-se que os três algoritmos demonstram seus melhores casos na criptografia de um pacote de imagens.

Os algoritmos que tiveram os piores desempenhos de tempo de acordo com este tamanho de chave foram o RC4 (em um dos casos o RC5 teve o mesmo desempenho), o AES e o Blowfish.

Apesar do algoritmo DES ser o único que possui uma chave de tamanho inferior ao dos demais, os testes realizados com ele foram comparados aos testes realizados nos demais algoritmos simétricos com o menor tamanho de chave (128 ***bits***).

Apesar de possuir uma chave com a metade do tamanho dos demais ele não apresentou um resultado de tempo decorrido satisfatório em comparação aos outros. Sendo assim, além de inviável sua utilização pelo fato de já ter sido quebrado por "força-bruta", ele não demonstra bons resultados.

Nos testes de integridade de imagem, no caso da imagem decriptografada, todos os algoritmos demonstraram o mesmo desempenho, não houve perdas nem ganhos de ***pixels*** nas imagens, as mesmas continuaram com o mesmo tamanho em ***bytes*** e a mesma altura (h) e largura (l) que a imagem original ao final do processo.

Na criação da imagem criptografada o algoritmo que demonstrou melhor desempenho foi o RC4, pois não houve ganhos de ***bytes*** no momento da criptografia – teve este desempenho por trabalhar com fluxo ao invés de blocos. Neste quesito os outros algoritmos testados apresentaram resultados parecidos, mas ainda assim satisfatórios, sendo passível a utilização de qualquer um deles.

A tabela abaixo demonstra as características e os resultados obtidos nos testes de integridade de imagem e de desempenho de tempo decorrido, realizados neste trabalho, com os algoritmos acima mencionados.

Quadro 24 - Caracteristicas e resultados obtidos com os algoritmos

Características	**Algoritmos**						
	AES	**BLOWFISH**	**RC2**	**RC5**	**RC4**	**DES**	**3DES**
Criptografia	Melhor: chave de 128 ***bits***	Melhor: chave de 192 ***bits***	Melhor: chave de 192 ***bits***	Melhor: chave de 128 ***bits***	Melhor: chave de 128 ***bits***	--	--
Decriptografia	Melhor: chave de 128 ***bits***	Melhor: chave de 128 ***bits***	Melhor: chave de 128 ***bits***	Melhor: chave de 192 ***bits***	Melhor: chave de 128 ***bits***	--	--
Eficiência – tempo decorrido	Melhor em 6 e pior em 6 dos 30 testes	Pior em 5 dos 30 testes	Pior em 3 dos 30 testes	Melhor em 20 e pior em 1 dos 30 testes	Melhor em 6 e pior em 5 dos 30 testes	Pior em 4 dos 10 testes	Pior em 8 dos 10 testes
Segurança	Chave de 256 ***bits***	Chave de 448 ***bits***	Chave de 1024 ***bits***	Chave de 2048 ***bits***	Chave de 256 ***bits***	Já foi quebrado por força bruta	Já foi quebrado por força bruta
Chave 64 ***bits*** Melhor em	--	--	--	--	--	Pacote de Imagens	--
Chave 128 ***bits*** Melhor em	Pacote de imagens	Pacote de imagens	Pacote de imagens	Pacote de imagens	Pacote de Imagens	--	--
Chave 192 ***bits*** Melhor em	Pacote de imagens	Pacote de imagens	Imagem-a-Imagem	Imagem-a-imagem	Imagem-a-imagem	--	Pacote de Imagens
Chave 256 ***bits*** Melhor em	Pacote de imagens	Imagem-a-Imagem	Pacote de imagens	Pacote de imagens	Pacote de imagens	--	--
Imagem Criptografada	Acréscimo de 16 ***bytes***	Acréscimo de 8 ***bytes***	Acréscimo de 8 ***bytes***	Acréscimo de 7 ***bytes***	Não há acréscimo de ***bytes***	Acréscimo de 8 ***bytes***	Acréscimo de 7 ***bytes***
Imagem Decriptografada	Não há acréscimo de ***bytes***	Não há acréscimo de ***bytes***	Não há acréscimo de ***bytes***	Não há acréscimo de ***bytes***	Não há acréscimo de ***bytes***	Não há acréscimo de ***bytes***	Não há acréscimo de ***bytes***

Apesar do algoritmo AES ser o padrão de segurança utilizado pelo governo dos Estados Unidos, ele não se sobressaiu ao algoritmo RC5 em nenhum dos três tamanhos de chaves, então conclui-se que na maioria dos casos de tempo decorrido na criptografia de imagens, não é aconselhável sua utilização.

Com o algoritmo assimétrico, não foi possível realizar o conjunto de testes pois o RSA retornava que o pacote não poderia ter uma quantidade superior a X ***bits***, e o tamanho da imagem ultrapassava este valor. Sendo assim, conclui-se que por trabalhar com uma quantidade pequena de ***bits***, a utilização deste algoritmo para criptografia de imagens criptografando os ***bits*** dos ***bytes*** da imagem não é possível.

De acordo com o que foi descrito, este trabalho conclui que o melhor algoritmo a ser utilizado na criptografia de imagens de radares de trânsito (no quesito tempo decorrido), com as dimensões e as características já descritas, é o RC5 – se comparado aos demais, este foi o algoritmo que apresentou o melhor desempenho na maioria dos casos.

Como a teoria afirma que quanto maior o tamanho da chave, maior a segurança, então, a utilização deste algoritmo em seu melhor caso é na criptografia de um pacote de X imagens (onde X é a quantidade) com uma chave de tamanho maior.

No quesito integridade de imagem, este trabalho conclui que o melhor algoritmo a ser utilizado é o RC4, pois mantem o mesmo tamanho em ***bytes*** e a mesma dimensão das imagens, tanto na função de criptografia, quanto na função de decriptografia.

Como foi descrito no capítulo 4.9, a utilização do algoritmo RSA na pratica é dada em conjunto com algum algoritmo simétrico. Então, conclui-se com este trabalho que o melhor caso para se criptografar imagens é: gerar a chave secreta com o algoritmo RC5 ou o RC4; criptografar o pacote de imagens com o algoritmo simétrico escolhido.

Gerar a chave pública e a chave privada com o algoritmo RSA; criptografar a chave gerada com o algoritmo simétrico escolhido, com a chave pública gerada no RSA; encaminhar o pacote criptografado com o algoritmo simétrico e a chave criptografada com o RSA pelo canal de comunicação inseguro.

Decriptografar a chave gerada com o algoritmo simétrico por meio da chave privada criada com o RSA e decriptografar o pacote de imagens com esta chave decriptografada.

Utilizando-se deste processo é possível alcançar os benefícios dos dois tipos de criptografia: a segurança das chaves encontrada nos algoritmos assimétricos, a agilidade e a possibilidade de criptografar grandes quantidades de ***bits*** encontrada nos algoritmos simétricos.

Além disso, pode ser utilizada em conjunto a assinatura digital para confirmar que certa imagem foi enviada pelo radar X, bem como, pode-se confirmar a integridade das imagens recebidas por meio de funções de ***hash***.

6.3 TRABALHO FUTURO

Este trabalho comparou tempo e integridade na geração de imagens criptografadas em extensão JPG, em um trabalho futuro é interessante verificar qual seria o tempo e a integridade da imagem criptografada se forem gerados arquivos em outras extensões (por exemplo: arquivos de texto) com todos os algoritmos testados (utilizando a ***Cipher***) e compará-los aos resultados apresentados neste trabalho.

Como foram utilizadas as classes padrões do Java para criptografar os ***bits*** dos ***bytes*** das imagens, e tomando como base os artigos apresentados por Silva et al: “Aplicação de técnicas de criptografia de chaves assimétricas em imagens” e “Criptografia assimétrica de imagens utilizando algoritmo RSA”.

Estes que abordam a criptografia das bandas de RGB, um outro item ao trabalho futuro é criptografar as bandas (com todos os algoritmos testados neste trabalho) e comparar o desempenho destes com os resultados obtidos neste trabalho.

Tendo em vista que o escopo deste trabalho analisava o teste do desempenho dos algoritmos simétricos e assimétrico, sem implementar nenhuma função para assinatura ou certificação digital, outro item possível ao trabalho futuro é a implementação destes para garantir que a imagem foi enviada por determinado radar e que a mesma continua integra.

Outro item para um futuro trabalho, é o teste da segurança dos algoritmos na criptografia específica de imagens, onde seriam aplicados alguns tipos de ataques à imagem criptografada e concluindo qual o nível de segurança dos algoritmos dado um determinado tamanho de chaves.

Como neste trabalho foram comparados somente os algoritmos de criptografia sem testar o meio de comunicação, será interessante simular a transmissão da imagem criptografada e das senhas no canal inseguro, criptografando no ponto X, enviando por uma rede cabeada, bem como por ***wireless***, e decriptografando no ponto Y. Testando assim o desempenho nos dois principais meios de comunicação.

REFERÊNCIAS

BHASKAR, D. Java – encryption and decryption of an image using blowfish algoritm. The Insane Techie, 2013. Disponível em: <http://www.theinsanetechie.in/2013/08/java-encryption-and-decryption-of-image.html>. Acesso em: 04 set. 2015.

BRAZIL, G. W. **Protegendo redes ad hoc com certificados digitais e limite criptográfico**. 2007. 109 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação, Universidade Federal Fluminense. Niterói, 2007.

BUGATTI, H. P. **Implementação e avaliação do algoritmo blowfish em c, java e microcontroladores pic**. 2005. 145 f. Monografia (Graduação) – Bacharelado em Ciência da Computação, Centro Universitário Eurípides de Marília. Marília, 2005.

CASTELLÓ, T; Vaz, V. Assinatura digital. Grupo de Teleinformática e Automação – Universidade Federal do Rio de Janeiro. Disponível em: <http://www.gta.ufrj.br/grad/07_1/ass-dig/TiposdeCriptografia.html#Topic6>. Acesso em: 11 ago. 2015

CENTRO DE ESTUDOS, RESPOSTA E TRATAMENTO DE INCIDENTES DE SEGURANÇA NO BRASIL. **Cartilha de Segurança Para Internet**. 2. ed. São Paulo: Comitê Gestor da Internet no Brasil, 2012.

CENTRO DE ESTUDOS, RESPOSTA E TRATAMENTO DE INCIDENTES DE SEGURANÇA NO BRASIL. Estatísticas dos incidentes reportados ao CERT.br. CERT BR, 2015. Disponível em: <http://www.cert.br/stats/incidentes/>. Acesso em: 12 ago. 2015.

FERREIRA, A. C. R. **Protocolos de segurança em redes sem fios**. 2006. 81 f. Tese (Mestrado) – Departamento de Matemática Pura, Faculdade de Ciências da Universidade do Porto. Portugal, 2006.

FILHO, M. O; NETO, V. H. **Processamento Digital De Imagens**. 1. ed. Rio de Janeiro: Brasport, 1999.

FONSECA, N. Java Cryptography Architecture e Java Cryptography Extension. Natanael Fonseca, 2010. Disponível em:

<http://www.natanaelfonseca.com.br/2010/10/java-cryptography-architecture-e-java.html>. Acesso em: 12 nov. 2015.

LAZANHA, F. **Implementação e desempenho do algortimo criptográfico "serpent"**. 2005. 86 f. Trabalho de Conclusão de Curso (Graduação) – Bacharelado em Ciência da Computação, Centro Universitário Eurípides de Marilia. Marília, 2005.

MEDEIROS, H. Criptografia assimétrica: criptografando e descriptografando dados em java. DevMedia, 2014. Disponível em: <http://www.devmedia.com.br/criptografia-assimetrica-criptografando-e-descriptografando-dados-em-java/31213>. Acesso em: 16 set. 2015.

MEDEIROS, H. Utilizando criptografia simétrica em java. DevMedia, 2014. Disponível em: <http://www.devmedia.com.br/utilizando-criptografia-simetrica-em-java/31170>. Acesso em: 02 set. 2015.

MENDES, R. D. **Programação Java Com Ênfase Em Orientação A Objetos**. 1. ed. São Paulo: Novatec, 2009.

MING, H. S. Fiscalização eletrônica do trânsito. Sinal de Trânsito, 2006. Disponível em: <http://www.sinaldetransito.com.br/artigos_area.php?tipo=tecnologia>. Acesso em: 25 ago. 2015.

MIRANDA, A. R. **Criptossistemas baseados em curvas elípticas**. 2002. 86 f. Dissertação (Mestrado) – Instituto de Computação, Universidade Estadual de Campinas. Campinas, 2002.

MORAES, F. R. Construção de um ambiente web com ferramentas para estudo de algoritmos de criptografia através do matlab. Rio de Janeiro: UFRJ, 2004. Disponível em:<http://www.ravel.ufrj.br/sites/ravel.ufrj.br/files/publicacoes/projetosfinais/pf_rosane_webenv_2004-06-24.pdf>. Acesso em: 18 ago. 2015.

MOREIRA, F. V. **Criptografia para dispositivos móveis**. 2010. 120 f. Trabalho de Graduação – Tecnologia em Banco de Dados, Faculdade de Tecnologia de São José dos Campos. São José dos Campos, 2010.

NAKAMURA, T. E; GEUS, L. P. **Segurança De Redes Em Ambientes Cooperativos**. 1. ed. São Paulo: Novatec, 2007.

ORACLE. Java™ platform, standard edition 8 API specification. Disponível em: <http://docs.oracle.com/javase/8/docs/api/>. Acesso em: 06 set. 2015.

OLIVEIRA, R. R. Criptografia simétrica e assimétrica: os principais algoritmos de cifragem. **Revista Segurança Digital**, [on-line], v. 05, p. 11-15, mar. 2012; continuação: v. 06, p. 21-24, mai. 2012. Disponível em <http://www.segurancadigital.info/>. Acesso em: 17 ago. 2015.

PAULINO, V. R. **Api java para a persistência e criptografia de arquivos de configuração xml e properties**. 2009. 71 f. Monografia (Especialização) – Programa de Pós-Graduação em Especialização em Tecnologia Java, Universidade Tecnológica Federal do Paraná. Curitiba, 2009.

SILVA, M. B. **Uma abordagem de infra-estrutura de chaves públicas para ambientes corporativos**. 2004. 161 f. Trabalho Final (Graduação) – Engenharia da Computação, Centro Universitário de Brasília. Brasília, 2004.

SILVA, B; ALMEIDA, P. H; OLIVEIRA, C. C; OLIVEIRA, C. D. Aplicação de técnicas de criptografia de chaves assimétricas em imagens. In SIBGRAPI – Conference on Graphics, Patterns and Images. 26. 2013. Arequipa, Peru. Disponível em: <http://www.ucsp.edu.pe/sibgrapi2013/eproceedings/wuw/114922_1.pdf>. Acesso em: 26 ago. 2015.

SILVA, B; ALMEIDA, P. H; OLIVEIRA, C. C; OLIVEIRA, C. D. Criptografia assimétrica de imagens utilizando algoritmo RSA. In Conferência de Estudos em Engenharia Elétrica. 7. 2013. Uberlândia. Disponível em: <http://www.ceel.eletrica.ufu.br/artigos2013/ceel2013_029.pdf>. Acesso em: 28 ago. 2015.

SOUSA, C. R. **Criptografia de chave pública:** algoritmos que possibilitam a criação de chave assimétrica. 2005. 13 f. Monografia (Graduação) – Bacharelado em Matemática, Universidade Católica de Brasília. Brasília, 2005.

SOUZA, M. X. T; SILVA, P. S. A. Criptografia em redes de computadores. Artigos Seinpar. Paranavaí: UNIPAR, 2013. Disponível em: <http://web.unipar.br/~seinpar/2013/artigos/Tiago%20Menezes%20Xavier%20de%20Souza.pdf>. Acesso em: 17 ago. 2015.

STALLINGS, W; BROWN, L. **Segurança De Computadores: Princípios e Práticas**. 2. ed. Tradução: Arlete Simille Marques. Rio de Janeiro: Elsevier, 2014.

TREVISAN, F. D; SACHI, P. S. R; SANABRIA, L. Estudo do padrão avançado de criptografia aes – advanced encryption standard. **Revista de Informática Teórica e Aplicada (RITA)**, [on-line], v. 20, n. 1, p. 13-24, 2013. Disponível em: <http://seer.ufrgs.br/index.php/rita/index>. Acesso em: 18 set. 2015.

UTILIZANDO Criptografia Em Java (Simétrica). 4Java, 2015. Disponível em: <http://4java.com.br/utilizando-criptografia-em-java-simetrica/>. Acesso em: 14 ago. 2015.

APÊNDICE A – Classes Java

Classe EncriptaDecripta3DES.java

```
public class EncriptaDecripta3DES {
   KeyGenerator keygenerator = null;
   Cipher cifraTDES = null;
   SecretKey chaveTDES = null;
   IvParameterSpec vetorI;
   SecretKey chaveencriptacao;
   static String IV = "JWAPYLMD";

public EncriptaDecripta3DES() {
      keygenerator = KeyGenerator.getInstance("DESede");
      chaveTDES = keygenerator.generateKey();
      cifraTDES = Cipher.getInstance("DESede/CBC/PKCS5Padding");
      vetorI = new IvParameterSpec(IV.getBytes("UTF-8"));
}

public void encrypt(String srcPath, String destPath) {
      File rawFile = new File(srcPath);
      File imagemEncriptada = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cifraTDES.init(Cipher.ENCRYPT_MODE, chaveTDES, vetorI);
      inStream = new FileInputStream(rawFile);
      outStream = new FileOutputStream(imagemEncriptada);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cifraTDES.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cifraTDES.doFinal());
      inStream.close();
      outStream.close();
   }

public void decrypt(String srcPath, String destPath) {
      File encryptedFile = new File(srcPath);
      File decryptedFile = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cifraTDES.init(Cipher.DECRYPT_MODE, chaveTDES, vetorI);
      inStream = new FileInputStream(encryptedFile);
      outStream = new FileOutputStream(decryptedFile);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cifraTDES.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cifraTDES.doFinal());
      inStream.close();
      outStream.close();
   }
}
```

Classe EncriptaDecriptaAES.java

```
public class EncriptaDecriptaAES {
   KeyGenerator keygenerator = null;
   Cipher cifraAES = null;
   SecretKey chaveAES = null;
   IvParameterSpec vetorI;
   SecretKey chaveencriptacao;
   static String IV = "AIULKWHXCMSNLRPZ";

public EncriptaDecriptaAES(int tamanhoC) {
      keygenerator = KeyGenerator.getInstance("AES");
      keygenerator.init(tamanhoC);
      chaveAES = keygenerator.generateKey();
      cifraAES = Cipher.getInstance("AES/CBC/PKCS5Padding");
      vetorI = new IvParameterSpec(IV.getBytes("UTF-8"));
   }

public void encrypt(String srcPath, String destPath) {
      File rawFile = new File(srcPath);
      File imagemEncriptada = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cifraAES.init(Cipher.ENCRYPT_MODE, chaveAES, vetorI);
      inStream = new FileInputStream(rawFile);
      outStream = new FileOutputStream(imagemEncriptada);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cifraAES.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cifraAES.doFinal());
      inStream.close();
      outStream.close();
   }

public void decrypt(String srcPath, String destPath) {
      File encryptedFile = new File(srcPath);
      File decryptedFile = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cifraAES.init(Cipher.DECRYPT_MODE, chaveAES, vetorI);
      inStream = new FileInputStream(encryptedFile);
      outStream = new FileOutputStream(decryptedFile);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cifraAES.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cifraAES.doFinal());
      inStream.close();
      outStream.close();
   }
}
```

Classe EncriptaDecriptaBlowfish.java

```
public class EncriptaDecriptaBlowfish {
   KeyGenerator keyGenerator = null;
   SecretKey secretKey = null;
   Cipher cipher = null;
   IvParameterSpec vetorI;
   static String IV = "AAAAAAAA";

public EncriptaDecriptaBlowfish(int tamanhoC) {
      keyGenerator = KeyGenerator.getInstance("Blowfish");
      keyGenerator.init(tamanhoC);
      secretKey = keyGenerator.generateKey();
      cipher = Cipher.getInstance("Blowfish/CBC/PKCS5Padding");
      vetorI = new IvParameterSpec(IV.getBytes("UTF-8"));
   }

void encrypt(String srcPath, String destPath) {
      File rawFile = new File(srcPath);
      File imagemEncriptada = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cipher.init(Cipher.ENCRYPT_MODE, secretKey, vetorI);
      inStream = new FileInputStream(rawFile);
      outStream = new FileOutputStream(imagemEncriptada);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cipher.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cipher.doFinal());
      inStream.close();
      outStream.close();
   }

void decrypt(String srcPath, String destPath) {
      File encryptedFile = new File(srcPath);
      File decryptedFile = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cipher.init(Cipher.DECRYPT_MODE, secretKey, vetorI);
      inStream = new FileInputStream(encryptedFile);
      outStream = new FileOutputStream(decryptedFile);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cipher.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cipher.doFinal());
      inStream.close();
      outStream.close();
   }
}
```

Classe EncriptaDecriptaDES.java

```
public class EncriptaDecriptaDES {
   KeyGenerator keygenerator = null;
   Cipher cifraDES = null;
   SecretKey chaveDES = null;
   IvParameterSpec vetorI;
   SecretKey chaveencriptacao;
   static String IV = "CMSNLRPZ";

public EncriptaDecriptaDES() {
      keygenerator = KeyGenerator.getInstance("DES");
      chaveDES = keygenerator.generateKey();
      cifraDES = Cipher.getInstance("DES/CBC/PKCS5Padding");
      vetorI = new IvParameterSpec(IV.getBytes("UTF-8"));
   }

public void encrypt(String srcPath, String destPath) {
      File rawFile = new File(srcPath);
      File imagemEncriptada = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cifraDES.init(Cipher.ENCRYPT_MODE, chaveDES, vetorI);
      inStream = new FileInputStream(rawFile);
      outStream = new FileOutputStream(imagemEncriptada);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cifraDES.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cifraDES.doFinal());
      inStream.close();
      outStream.close();
   }

public void decrypt(String srcPath, String destPath) {
      File encryptedFile = new File(srcPath);
      File decryptedFile = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cifraDES.init(Cipher.DECRYPT_MODE, chaveDES, vetorI);
      inStream = new FileInputStream(encryptedFile);
      outStream = new FileOutputStream(decryptedFile);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cifraDES.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cifraDES.doFinal());
      inStream.close();
      outStream.close();
   }
}
```

Classe EncriptaDecriptaRC2.java

```
public class EncriptaDecriptaRC2 {
   KeyGenerator keyGenerator = null;
   SecretKey secretKey = null;
   Cipher cipher = null;
   IvParameterSpec vetorI;
   static String IV = "AAAAAAAA";

public EncriptaDecriptaRC2(int tamanhoC) {
      keyGenerator = KeyGenerator.getInstance("RC2");
      keyGenerator.init(tamanhoC);
      secretKey = keyGenerator.generateKey();
      cipher = Cipher.getInstance("RC2/CBC/PKCS5Padding");
      vetorI = new IvParameterSpec(IV.getBytes("UTF-8"));
   }

void encrypt(String srcPath, String destPath) {
      File rawFile = new File(srcPath);
      File imagemEncriptada = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cipher.init(Cipher.ENCRYPT_MODE, secretKey, vetorI);
      inStream = new FileInputStream(rawFile);
      outStream = new FileOutputStream(imagemEncriptada);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cipher.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cipher.doFinal());
      inStream.close();
      outStream.close();
   }

void decrypt(String srcPath, String destPath) {
      File encryptedFile = new File(srcPath);
      File decryptedFile = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cipher.init(Cipher.DECRYPT_MODE, secretKey, vetorI);
      inStream = new FileInputStream(encryptedFile);
      outStream = new FileOutputStream(decryptedFile);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cipher.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cipher.doFinal());
      inStream.close();
      outStream.close();
   }
}
```

Classe EncriptaDecriptaRC4.java

```
public class EncriptaDecriptaRC4 {
   KeyGenerator keyGenerator = null;
   SecretKey secretKey = null;
   Cipher cipher = null;

public EncriptaDecriptaRC4(int tamanhoC) {
       keyGenerator = KeyGenerator.getInstance("RC4");
       keyGenerator.init(tamanhoC);
       secretKey = keyGenerator.generateKey();
       cipher = Cipher.getInstance("RC4");
   }

void encrypt(String srcPath, String destPath) {
       File rawFile = new File(srcPath);
       File imagemEncriptada = new File(destPath);
       InputStream inStream = null;
       OutputStream outStream = null;
       cipher.init(Cipher.ENCRYPT_MODE, secretKey);
       inStream = new FileInputStream(rawFile);
      outStream = new FileOutputStream(imagemEncriptada);
       byte[] buffer = new byte[1024];
       int len;
       while ((len = inStream.read(buffer)) > 0) {
          outStream.write(cipher.update(buffer, 0, len));
          outStream.flush();
       }
       outStream.write(cipher.doFinal());
       inStream.close();
       outStream.close();
   }

void decrypt(String srcPath, String destPath) {
       File encryptedFile = new File(srcPath);
       File decryptedFile = new File(destPath);
       InputStream inStream = null;
       OutputStream outStream = null;
       cipher.init(Cipher.DECRYPT_MODE, secretKey);
       inStream = new FileInputStream(encryptedFile);
       outStream = new FileOutputStream(decryptedFile);
       byte[] buffer = new byte[1024];
       int len;
       while ((len = inStream.read(buffer)) > 0) {
          outStream.write(cipher.update(buffer, 0, len));
          outStream.flush();
       }
       outStream.write(cipher.doFinal());
       inStream.close();
       outStream.close();
   }
}
```

Classe EncriptaDecriptaRC5.java

```
public class EncriptaDecriptaRC5 {
   KeyGenerator keyGenerator = null;
   SecretKey secretKey = null;
   Cipher cipher = null;
   static String IV = "LKPEJRHC";
   IvParameterSpec vetorI;

public EncriptaDecriptaRC5(int tamanhoC) {
       Security.addProvider(new BouncyCastleProvider());
       keyGenerator = KeyGenerator.getInstance("RC5");
       keyGenerator.init(tamanhoC);
       secretKey = keyGenerator.generateKey();
       cipher = Cipher.getInstance("RC5/CBC/PKCS5Padding");
       vetorI = new IvParameterSpec(IV.getBytes("UTF-8"));
   }

void encrypt(String srcPath, String destPath) {
       File rawFile = new File(srcPath);
       File imagemEncriptada = new File(destPath);
       InputStream inStream = null;
       OutputStream outStream = null;
       cipher.init(Cipher.ENCRYPT_MODE, secretKey, vetorI);
       inStream = new FileInputStream(rawFile);
       outStream = new FileOutputStream(imagemEncriptada);
       byte[] buffer = new byte[1024];
       int len;
       while ((len = inStream.read(buffer)) > 0) {
          outStream.write(cipher.update(buffer, 0, len));
          outStream.flush();
       }
       outStream.write(cipher.doFinal());
       inStream.close();
       outStream.close();
   }

void decrypt(String srcPath, String destPath) {
       File encryptedFile = new File(srcPath);
       File decryptedFile = new File(destPath);
       InputStream inStream = null;
       OutputStream outStream = null;
       cipher.init(Cipher.DECRYPT_MODE, secretKey, vetorI);
       inStream = new FileInputStream(encryptedFile);
       outStream = new FileOutputStream(decryptedFile);
       byte[] buffer = new byte[1024];
       int len;
       while ((len = inStream.read(buffer)) > 0) {
          outStream.write(cipher.update(buffer, 0, len));
          outStream.flush();
       }
       outStream.write(cipher.doFinal());
       inStream.close();
       outStream.close();
   }
}
```

Classe EncriptaDecriptaRSA.java

```
public class EncriptaDecriptaRSA {
   KeyPairGenerator key;
   KeyPair par;
   PublicKey chavePublica;
   PrivateKey chavePrivada;
   Cipher cifra;

public EncriptaDecriptaRSA(tamanhoC) {
      key = key.getInstance("RSA");
      key.initialize(tamanhoC);
      par = key.generateKeyPair();
      chavePublica = par.getPublic();
      chavePrivada = par.getPrivate();
      cifra = Cipher.getInstance("RSA");
   }

public void encrypt(String srcPath, String destPath) {
      File rawFile = new File(srcPath);
      File imagemEncriptada = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cifra.init(Cipher.ENCRYPT_MODE, chavePublica);
      inStream = new FileInputStream(rawFile);
      outStream = new FileOutputStream(imagemEncriptada);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cifra.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cifra.doFinal());
      inStream.close();
      outStream.close();
   }

public void decrypt(String srcPath, String destPath) {
      File rawFile = new File(srcPath);
      File imagemEncriptada = new File(destPath);
      InputStream inStream = null;
      OutputStream outStream = null;
      cifra.init(Cipher.DECRYPT_MODE, chavePrivada);
      inStream = new FileInputStream(rawFile);
      outStream = new FileOutputStream(imagemEncriptada);
      byte[] buffer = new byte[1024];
      int len;
      while ((len = inStream.read(buffer)) > 0) {
         outStream.write(cifra.update(buffer, 0, len));
         outStream.flush();
      }
      outStream.write(cifra.doFinal());
      inStream.close();
      outStream.close();
```