Mário Luiz Rodrigues Oliveira

Uma Análise da Segurança e da Eficiência do Algoritmo de Criptografia Posicional

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Orientador
Professor MSc. Antonio Maria Pereira de Resende

Co-Orientador
Professor DSc. Lucas Monteiro Chaves

Lavras
Minas Gerais - Brasil
2002

Mário Luiz Rodrigues Oliveira

Uma Análise da Segurança e da Eficiência do Algoritmo de Criptografia Posicional

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em *27 de março de 2002*

Professor MSc. Bruno de Oliveira Schneider

Professor MSc. Antonio Maria Pereira de Resende
(Orientador)

Professor DSc. Lucas Monteiro Chaves
(Co-Orientador)

Lavras
Minas Gerais - Brasil

A
minha família,
In memorian de minha querida avó Josefina
por todo o seu amor, carinho e afeto para comigo.
In memorian de Olímpia (Dinha).

Agradecimentos

Agradeço a Deus.
Agradeço aos meus pais, pelo apoio e incentivo durante esses anos de estudo.
Agradeço ao Flávio e a Ana Paula, meus queridos irmãos, por todo o seu amor e amizade para comigo.
Agradeço aos professores Antonio Maria e Lucas pelo incentivo e orientação deste trabalho.
Aos moradores do apartamento 304 (Túlio, Adriano, Tiago, Sidney, Erasmo) pelos bons momentos ali vividos e pela grande amizade formada durante esses anos de convívio.
A Diego e Sérgio pelas críticas e sugestões a este trabalho.
E um agradecimento especial à querida Nega por todo o seu carinho e disposição em ajudar.

Resumo

Este trabalho objetivou analisar o algoritmo de criptografia posicional quanto a sua eficiência e segurança. Para tal propósito implementou-se o algoritmo posicional em C++ afim de medir o tempo gasto no processo de codificação. Para obter um parâmetro comparativo de sua eficiência implementou-se o algoritmo de criptografia RSA. Avaliou-se a segurança do algoritmo posicional verificando-se a influência da operação matemática *mod* na construção das chaves de codificação e a possibilidade de decifrá-lo através dos métodos da força bruta e de análise de freqüência de símbolos.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O crescimento contínuo da Internet e das redes privadas de computadores traz consigo o desafio de garantir-se a segurança das informações que trafegam por essas redes. Assim a necessidade de prover a segurança dessas informações torna-se cada vez mais importante à medida que aumenta a informatização da sociedade atual e cresce o valor agregado às informações disponíveis nas redes de computadores.

Dessa forma necessita-se de maneiras eficazes que garantam essa segurança e dentre essas maneiras pode-se citar a criptografia, objeto de estudo deste trabalho.

Numa sociedade onde é crescente o uso da informática para a realização de transações financeiras e comerciais através de redes de computadores inseguras, a criptografia é de vital importância. Considere por exemplo uma transação comercial na qual passam-se informações tais como o número do cartão de crédito com sua respectiva senha, ou uma transação bancária onde estarão disponíveis o número da conta do usuário e sua senha. Se tais informações trafegam por uma rede desprotegida de criptografia é possível lê-las. As conseqüências deste fato podem ser muito desagradáveis para algumas pessoas dependendo do conteúdo da mensagem.

Este trabalho enfocou principalmente o estudo do Algoritmo de Criptografia Posicional [2], avaliando-o nos quesitos:

- eficiência e
- segurança.

A eficiência do algoritmo será considerada apenas em função da complexidade temporal e velocidade de execução do mesmo. Para tal propósito aplicou-se

os conceitos da teoria de complexidade computacional de algoritmos usando a notação O(n) [7] e verificou-se a eficiência na prática medindo-se o tempo gasto pelo algoritmo. Para fazer comparativamente uma análise da eficiência do algoritmo posicional, implementou-se também uma versão do algoritmo RSA com chaves de 16 bits.

Quanto a segurança do algoritmo posicional verificou-se a influência da operação matemática *mod* na construção das chaves, pois esta operação reduz o universo de chaves a serem usadas e isso a princípio tornaria possível decifrar o código num tempo satisfatório. Analisou-se também a possibilidade de decifrar o algoritmo posicional através dos métodos da força bruta e de análise de freqüência.

Os temas abordados neste trabalho estão estruturados da seguinte forma: no capítulo 2 apresenta-se uma revisão de literatura sobre criptografia abordando conceitos gerais e os fundamentos matemáticos necessários ao entendimento dos métodos de criptografia posicional e RSA, a seguir descreve-se o algoritmo RSA , o algoritmo posicional e noções de teoria de complexidade de algoritmos. No capítulo 3 é apresentada a avaliação do desempenho e segurança do algoritmo posicional. Faz-se uma discussão sobre os resultados obtidos na avaliação do desempenho e segurança do algoritmo posicional no capítulo 4 e, finalmente, nos capítulos 5 e 6 são apresentadas respectivamente a conclusão e as sugestões de trabalhos futuros.

Capítulo 2

Revisão de Literatura

2.1 Introdução

O presente capítulo tem por objetivo conceituar e apresentar um breve histórico sobre a criptografia, descrever sucintamente alguns métodos criptográficos tradicionais e computacionais, mostrar a importância da criptografia na sociedade moderna e as formas de ataque aos algoritmos criptográficos, além dos critérios utilizados para avaliar a segurança dos algoritmos criptográficos.

Apresenta-se também os conceitos matemáticos que constituem os pilares do algoritmo de criptografia RSA. Ressalta-se que a abordagem aqui adotada será a suficiente à compreensão RSA, sendo indicada uma bibliografia complementar aos interessados no assunto.

E por fim mostra-se o algoritmo de criptografia RSA, um método de chave pública ou assimétrica, criado por Rivest, Shamir e Adleman em 1977 e faz-se algumas considerações sobre sua segurança.

2.2 Conceitos e Histórico

A palavra criptografia é definida pelo dicionário Aurélio como “a arte de escrever em cifra ou em código”, significado este que remete ao grego *cryptos* que significa secreto, oculto. Em [6] encontra-se o modelo de criptosistema assim descrito: "dada uma mensagem e uma chave de codificação como entrada, o método de codificação produz como resultado uma mensagem codificada, a qual pode ser armazenada num meio qualquer ou enviada a um destinatário; para decodificar esta mensagem utiliza-se o método de decodificação passando como entradas a men-

sagem codificada e a chave de decodificação obtendo-se a mensagem original". Neste trabalho chama-se a mensagem original de texto simples e a mensagem codificada resultante do processo de codificação de texto codificado.

Uma complementação ao conceito de criptografia nos remete a sua finalidade, como visto em [1] "a criptografia estuda os métodos para codificar uma mensagem de modo que só seu destinatário legítimo [1] consiga interpretá-la, é a arte dos códigos secretos"e [8] "a criptografia concerne a construção de esquemas que devem ser capazes de resistir a qualquer tipo de abuso, tais esquemas são construídos de modo a manter uma funcionalidade desejada, mesmo sob tentativas maliciosas com a intenção de fazê-los desviar de sua funcionalidade recomendada".

Como visto acima, ao usar a criptografia para proteger informações, precisa-se ter em mente que pessoas com acesso não autorizado a tais informações estão registrando as informações criptografadas e tentam decifrá-las a força. Para perceber a importância de tal afirmação considere a opinião de Withfield Diffie, inventor da criptografia de chave pública: "se seus dados não estão sujeitos a este tipo de ataque, não é preciso criptografá-los".

Paralelamente à criptografia, mas trilhando o caminho inverso, caminha a criptoanálise, a qual objetiva decifrar os códigos secretos, sendo os profissionais que realizam tal tarefa denominados criptoanalistas. Nesse ponto cumpre salientar a diferença entre os termos decodificar e decifrar para evitar quaisquer ambigüidade quanto aos seus significados. Conforme [1] decodificar é o que um usuário legítimo [2] do código faz quando recebe um texto codificado e deseja vê-lo. E decifrar significa ler um texto codificado sem ser o usuário legítimo de tal texto, portanto para decifrar um texto é necessário *quebrar* o código.

Outro conceito de grande importância nos sistemas criptográficos é a divisão dos mesmos em criptografia de chave única ou simétrica e criptografia de chave pública ou assimétrica. No sistema de chave única ou simétrica utiliza-se uma única chave no processo de codificação e decodificação do texto, ou seja, a chave usada para codificar o texto é a mesma usada na decodificação. Contrapondo-se a esse conceito, tem-se o sistema de chave pública ou assimétrica, no qual utilizam-se duas chaves: uma chamada de *chave privada* e que deve ser mantida em segredo, a outra chamada de *chave pública* e como o próprio nome indica é de conhecimento público. A figura 2.1 mostra um sistema criptográfico de chave pública, enfatizando o uso de duas chaves diferentes (*Ke* e *Kd*) no processo de codificação.

Dados esses conceitos iniciais necessários à compreensão do tema aqui abor-

[1] Destinatário legítimo é a pessoa a quem está endereçada a mensagem

[2] Usuário legítimo é o emitente ou o destinatário de uma mensagem

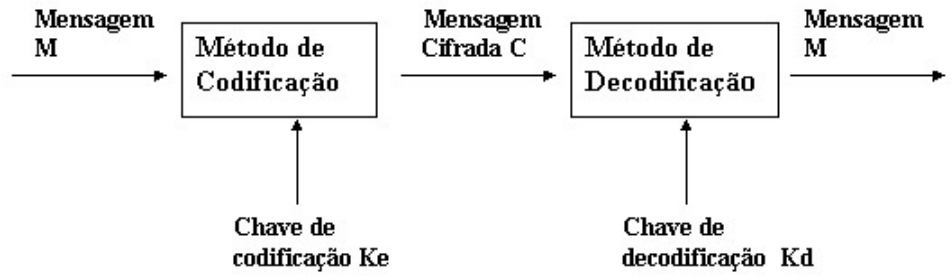

Figura 2.1: Sistema Criptográfico de Chave Pública

dado, pode-se iniciar um breve histórico sobre a criptografia.

A idéia inicial de criptografia pode ser percebida nos *hieróglifos* egípcios [1] e também entre generais gregos e romanos, os quais usavam a criptografia para enviar mensagens em códigos aos comandantes de campo [5]. Vê-se portanto que as raízes da criptografia são de longa data, o que permitiu a essa arte um grande desenvolvimento. Desenvolvimento este que segundo [9] deve-se historicamente a quatro categorias:

- militares;
- diplomatas;
- pessoas que gostam de guardar memórias;
- amantes.

E dentre as categorias acima, tiveram papel preponderante as organizações militares.

A longevidade da criptografia fez surgir uma classificação da mesma em *Criptografia Tradicional* e *Criptografia Moderna ou Computacional*.

2.2.1 A Criptografia Tradicional

Conforme visto em [6] antes da era computacional, a criptografia baseava-se na substituição de um caracter por outro ou na troca de posição dos caracteres no texto, sendo por isso denominada criptografia orientada a caracter. Para aumentar a segurança, alguns métodos utilizavam tanto a substituição quanto a troca de

posição dos caracter e de preferência várias vezes. É interessante notar que tais métodos são todos simétricos.

Cifras de Substituição

Os métodos de substituição trocam cada caracter no texto simples por outro caracter no texto codificado. Tal substituição é realizada para tornar o texto codificado mais obscuro e incompreensível. Para decodificar o texto faz-se o processo inverso, de forma a restaurar o texto simples. Segundo [6] são considerados somente os 26 caracteres que são letras. Segue abaixo alguns tipos de cifras de substituição catalogadas por [6].

- **Substituição monoalfabética:** neste tipo de criptografia cada caracter é substituído por outro de acordo com uma tabela ou regra simples. A cifra de substituição mais conhecida é a *Cifra de César*, na qual cada caracter é substituído por três caracteres adiante do alfabeto; assim A é substituído por D, B é substituído por E e assim sucessivamente até Z ser substituído por C. Para decodificar tal método é suficiente retroceder cada caracter do texto codificado em três posições.

  Uma generalização deste método pode ser obtida deslocando-se o alfabeto de uma constante K posições, sendo K a chave de criptografia a ser utilizada. Um exemplo dessa generalização é o algoritmo ROT13 onde os caracteres são avançados 13 posições. Outra alternativa para a substituição monoalfabética é utilizar um mapeamento individual de cada caracter em substituição a um deslocamento constante, porém neste caso necessita-se de uma tabela para indicar as substituições que estão sendo feitas.

- **Substituição monofônica:** trabalha de maneira à substituição monoalfabética, no entanto cada caracter do texto simples pode ser mapeado para um ou vários caracteres no texto codificado. Assim, A pode ser substituído por 10, 2, 5 e B pode ser substituído por 7, 16, 41.

- **Substituição polialfabética:** este método usa uma combinação de várias substituições monoalfabéticas, usadas em rotação de acordo com algum critério ou chave. Como exemplo, considere uma substituição polialfabética em que são utilizadas quatro tabelas, usadas em alternância a cada quatro caracteres: a primeira tabela seria usada para substituir os caracteres nas posições 1,5,9 e assim por diante; a segunda para substituir os caracteres nas posições 2,6,10 e assim por diante; a terceira substituiria os caracteres das

posições 3,7,11 e assim por diante e a quarta tabela substituiria os caracteres nas posições 4,8,12 e assim por diante. Um exemplo clássico deste tipo de cifra é a *Cifra de Vigenère* a qual é constituída por 26 Cifras de César, cada uma com um deslocamento diferente.

- **Substituição por polígramos:** neste tipo de substituição são utilizados grupos de caracteres em vez de trabalhar com caracteres individuais. Então poderia-se ter o grupo ABC substituído por RTQ ou ABB substituído por KSC.

Numa primeira análise tais métodos podem parecer seguros, no entanto tais cifras podem ser facilmente quebradas fazendo-se a análise de freqüência de cada caracter no texto codificado e comparando-se estas freqüências com aquelas que normalmente encontram-se na língua na qual o texto simples foi escrito. Weber [6] citando Tanenbaum afirma que a maioria das linguagens possui uma redundância tão alta que é necessária uma quantidade bem pequena de texto codificado para que se possa realizar a criptoanálise baseada em análise de freqüência.

Cifras de Transposição

Nós métodos que utilizam as Cifras de Transposição, cada caracter permanece inalterado, no entanto sua posição na no texto codificado é alterada de acordo com uma regra ou função. Um exemplo de criptografia usando-se tal método é considerar o texto simples como uma matriz de caracteres e tal texto codificado é a matriz transposta da matriz de caracteres original.

Para diferenciar entre um texto codificado por cifras de substituição de outro codificado por cifras de transposição utiliza-se da análise de freqüência dos caracteres, se a freqüência for a mesma da língua, tem-se uma transposição, caso contrário uma substituição [6].

2.2.2 A Criptografia Moderna ou Computacional

Atualmente com o grande avanço da tecnologia computacional procedeu-se a substituição dos métodos criptográficos tradicionais por métodos criptográficos computacionais, onde as operações são implementadas por um computador ou por um circuito integrado especial tendo como conseqüência a aceleração dos processos de codificação e decodificação. Tal aumento nas velocidades de codificação/decodificação fez-se com que diversos passos extras de substituição e transposição fossem usados, a fim de dificultar a criptoanálise.

Dessa forma deseja-se a obtenção de um método de codificação ideal e para tal o método deve garantir que a probabilidade de ocorrência de qualquer símbolo no texto codificado seja exatamente igual às probabilidades de todos os demais símbolos, ou seja, a freqüência dos símbolos é homogênea. Dessa forma garante-se que a alteração de um único caracter no texto simples tenha a probabilidade de alterar metade dos símbolos do texto codificado e vice-versa e com isso impede-se qualquer análise por freqüência dos símbolos [6].

2.3 Exemplos de Métodos Criptográficos

Esta seção apresentará alguns métodos criptográficos existentes, tendo o intuito de mostrar sucintamente algumas idéias utilizadas para criptografar dados.

Nesta apresentação divide-se os algoritmos criptográficos em algoritmos simétricos e algoritmos assimétricos. E iniciando esta descrição pelos algoritmos simétricos, tem-se:

- **DES:** O *DES*(Data Encryption Standard) é o exemplo mais difundido de algoritmo criptográfico de chave única [6]. Ele foi desenvolvido pela IBM e adotado como padrão nos Estados Unidos em 1977. O DES é um algoritmo de bloco e trabalha dividindo o texto simples em blocos de 8 caracteres, cifrando cada bloco com uma chave de 56 bits (mais 8 bits de paridade, totalizando uma chave de 64bits). Este método é passível de ser quebrado usando o método da força bruta, bastando para tal testar as 2^{56} chaves possíveis, no entanto, já foi demonstrado que é possível quebrá-lo utilizando-se o ataque do texto plano escolhido e adaptativo em 2^{47} tentativas ou até mesmo em 2^{43} [6]. Maiores detalhes sobre o DES podem ser encontrados em [6].

- **Triple-DES:** Este algoritmo é um método para tornar o DES mais seguro e para atingir tal objetivo aplica-se o algoritmo do DES três vezes com duas chaves diferentes: inicialmente cifra-se o texto com a chave C1, depois com a chave C2 e finalmente com a C1 novamente. Para quebrar tal método com o método da força bruta requer-se 2^{112} tentativas. Atualmente é usado por instituições financeiras como alternativa ao DES [5].

- **IDEA:** O algoritmo de criptografia de dados internacional - IDEA foi desenvolvido na Suíça por James Massey e Xuenjia Lai e publicado em 1990. Ele é um algoritmo de blocos com tamanho de 64 bits e utiliza chaves de 128 bits. Acredita-se que ele seja um algoritmo bastante poderoso, visto

que ainda não existe nenhum método de ataque efetivo contra o mesmo e também por ele tem resistido bem contra os métodos aplicados com êxito sobre outros algoritmos [6]. O uso generalizado de tal algoritmo foi bloqueado por uma série de patentes, atualmente em mãos da Ascom-Tech AG [5].

- **Blowfish:** Tal algoritmo de criptografia em bloco foi inventado por Bruce Schneier e permite a utilização de chaves de até 448 bits, sendo otimizado para ser executado em máquinas de 32 ou 64 bits.

Além desses algoritmos acima descritos podem ser citados: RC2, RC4, RC5, MMB, Lucifer, NewDES entre outros. Aos interessados podem consultar [6], artigo no qual são citados os algoritmos de criptografia mais comumentes encontrados na literatura com uma pequena descrição sobre os mesmo e com referências para os que buscam detalhes aprofundados sobre tais algoritmos.

Descritos alguns algoritmos simétricos, descreve-se alguns métodos de criptografia de chave pública.

- **Diffie-Hellman:** primeiro método de chave pública proposto tendo por objetivo resolver o problema de distribuição das chaves e não podendo ser utilizado para cifrar mensagens. Tal algoritmo baseia-se no problema do logaritmo discreto, o qual é NP completo. Maiores detalhes sobre tal algoritmo podem ser encontrados em Diffie e Hellman, citados por [6].

- **Rabin:** este método baseia-se na dificuldade de se extrair a raiz quadrada em aritmética modular de um número composto. Para aplicá-lo escolhe-se dois números primos, p e q, sendo que ambos devem ser congruentes a 3 módulo 4 e tais primos são a chave privada e a chave pública é o número n calculado como o produto de p e q. Em [6] podem ser encontrados mais detalhes sobre tal algoritmo.

- **ElGamal:** criado por Taher ElGamal é um sistema criptográfico de chave pública que pode ser usado tanto para cifrar mensagens quanto para assinatura digital. Tem sua segurança baseada na dificuldade de calcular logaritmos discretos e aritmética modular.

Além dos métodos acima citados, podem ser encontrados outros na literatura, tais como: DSA, LUC, DSS, Método da Mochila entre outros. Aos interessados recomenda-se pesquisar em [6] o qual indica uma série de algoritmos de criptografia de chave pública com uma sucinta descrição sobre os mesmo, além de indicar fontes onde podem ser encontrados tais algoritmos com riqueza de detalhes.

2.4 A Importância da Criptografia para a Sociedade Atual

A importância da criptografia na sociedade atual pode ser percebida através de alguns exemplos do uso de tal tecnologia. Recentemente encontra-se a criptografia protegendo dados confidenciais em transações comerciais ou bancárias, informações armazenadas em cartões inteligentes e transmissões de televisão via satélite [5]. Ao usar-se a criptografia com o objetivo de impedir o acesso às informações por usuários ilegítimos [3]diz-se que ela cumpriu a função de garantir a *confidencialidade*. Além dessa função pode-se relacionar mais três funções que a criptografia desempenha nos sistemas modernos de informações conforme visto em [5]:

- **autenticação:** o uso de assinaturas digitais permite a identificação do autor de uma mensagem e assim o destinatário da mensagem pode verificar a identidade do remetente;
- **integridade:** alguns métodos de assinatura digital permitem verificar se uma mensagem não foi modificada em trânsito;
- **não-repúdio:** pode-se utilizar a criptografia para criar recibos e assim o autor da mensagem não poderá negar falsamente que a tenha enviado.

2.5 Formas de Ataques a Métodos Criptográficos

Pessoas não autorizadas que tenham acesso ao modelo de criptosistema são chamadas de atacantes e em [6] os atacantes são divididos em duas categorias: atacantes ativos e atacantes passivos. Os atacantes ativos podem interceptar uma mensagem e alterá-la ou trocá-la por outra mensagem, ao passo que os atacantes passivos somente conseguem ter acesso a mensagens sem no entanto modificá-las.

Para atacar um sistema de criptografia, o criptoanalista pode agir das seguintes formas segundo Schneier citado por [6]:

- **ataque do texto cifrado:** neste tipo de ataque o criptoanalista tem a sua disposição uma grande quantidade de textos codificados, mas desconhece os textos simples e as chaves utilizadas e sua função é deduzir tais chaves;
- **ataque do texto conhecido:** aqui o criptoanalista tem a sua disposição uma grande quantidade de textos codificados, conhece também os textos simples

[3]Usuário ilegítimo é qualquer pessoa que tenha acesso à mensagem, sem no entanto ser o emitente ou destinatário da mesma

correspondentes e sua tarefa é deduzir as chaves utilizadas no processo de codificação. Na prática existem vários tipos de mensagens padrões como cabeçalhos de e-mail e arquivos, nomes de pessoas e mensagens de login, das quais pode-se extrair informações não codificadas.

- **ataque adaptativo do texto escolhido:** este tipo de ataque diferencia-se do anterior por permitir uma realimentação entre um texto escolhido para codificação e o próximo texto. No método anterior o criptoanalista somente era capaz de fornecer textos uma única vez. Neste ele pode alimentar o sistema com um pequeno texto, analisar os resultados e realimentar o sistema com outro pequeno texto. Dessa forma, este ataque mais efetivo do que o anterior. O objetivo aqui também é descobrir as chaves utilizadas.

- **ataque do texto cifrado escolhido:** neste tipo de ataque o criptoanalista não só conhece uma grande quantidade de textos simples e seus equivalentes codificados, mas também pode produzir um texto codificado específico para ser decodificado e obter o resultado produzido. Aqui também objetiva-se encontrar as chaves envolvidas no processo de codificação.

Em todos os tipos de ataques descritos acima presume-se que o criptoanalista tenha total conhecimento dos métodos de codificação e decodificação.

Além dos tipos de ataques enumerados pode-se citar o ataque da força bruta, na qual testa-se todas as chaves possíveis e dessa forma obterá sucesso com uma delas, desde que possa ser reconhecido o resultado da chave correta. Não é possível defender-se deste tipo de ataque, pois não pode-se evitar que um atacante tente decifrar uma mensagem testando todas as chaves possíveis, ressalta-se no entanto que este tipo de ataque é ineficiente visto a quantidade de chaves a serem testadas.

Estas são formas de atacar um sistema criptográfico e não a maneira usada para inferir a chave de criptografia. Como maneira para descobrir a chave pode-se citar a análise de freqüência, que constitui em analisar as freqüências do texto codificado; feito isso supõe-se que o caracter de maior freqüência no texto codificado corresponde ao código do caracter de maior freqüência na língua na qual foi escrito o texto simples e a partir dessa informação o criptoanalista pode tentar descobrir a chave usada na codificação.

2.6 Segurança dos Métodos Criptográficos

Segurança é um atributo muito difícil e complexo de ser implementado num sistema [6] e assim um sistema pode ser considerado seguro se ainda não foi possível

determinar uma maneira de torná-lo inseguro. Sendo esta opinião também compartilhada por [8] que afirma ser muito complicado avaliar a segurança de um método criptográfico.

Em [8] encontram-se duas abordagens sobre segurança em métodos criptográficos, a saber:

- **clássica:** nessa abordagem um método é considerado seguro apenas se a chave em uso é pelo menos tão longa quanto o comprimento do texto simples, tal abordagem é baseada na teoria da informação;
- **moderna:** abordagem baseada na complexidade computacional e considera seguro um sistema onde não é possível extrair informações eficientemente de um texto codificado , mesmo que o texto codificado contenha dados sobre o texto simples.

Em [6] citando Schneier encontra-se os seguintes requisitos para considerar robusto um método criptográfico:

- o criptoanalista tem acesso à descrição do algoritmo;
- o criptoanalista tem acesso a uma grande quantidade de textos codificados e as seus correspondentes textos simples;
- o criptoanalista é capaz de escolher quais mensagens serão cifradas e receber as mensagens cifradas correspondentes.

Considera-se inseguro o método que não atende às premissas acima citadas. Também percebe-se que atualmente não se julga segurança de um método criptográfico baseado na obscuridade do mesmo.

Finalmente considere a premissa de Kerckhoffs sobre a segurança dos métodos criptográficos: o mecanismo deve ser tão seguro que nem mesmo seu autor deve ser capaz de decifrar uma mensagem sem dispor da chave utilizada [6], ou ainda "a chave de criptografia pode cair nas mãos do inimigo sem inconvenientes"disse Kerckhoffs.

2.7 Fundamentos Matemáticos

2.7.1 Relações de Equivalência

A aritmética modular é o ramo da matemática dedicado ao estudo dos fenômenos cíclicos. Por exemplo, pode-se somar 15 + 13 e obter como resultado 4 e este

resultado está correto desde que se considere os números 15, 13 e 4 como horas, logo se são 15 horas e passam 13 horas, agora são 4 horas da manhã. Essa associação com o relógio é interessante e ajudará a compreender as propriedades desta aritmética.

Em [1], tem-se a introdução deste assunto utilizando a noção de relações de equivalência, método este que também será usado neste trabalho.

Em [1] vê-se que dado um conjunto **X**, o qual pode ser finito ou infinito, uma *relação* nesse conjunto **X** é definida dizendo-se como comparar dois elementos deste conjunto. Para tornar mais clara a idéia de relação considere os seguintes exemplos:

- No conjunto dos números inteiros, pode-se considerar duas relações naturais, a relação de igualdade e a relação de desigualdade;
- Num conjunto de bolas coloridas, pode-se considerar a relação de bolas de uma mesma cor.

Em ambos os exemplos define-se uma maneira de comparar os elementos de um conjunto, logo define-se uma relação neste conjunto. E tendo clara a noção de relação, pode-se conceituar *relações de equivalência*.

Dados um conjunto **X** e uma relação que será denotada por $\sim$ definida neste conjunto e considerando-se ainda os elementos x, y e z pertencentes ao conjunto **X**, define-se esta relação como uma *relação de equivalência* se as seguintes propriedades forem satisfeitas para quaisquer x, y e z:

- $x \sim x$ (reflexiva)
- se $x \sim y$ então $y \sim x$ (simétrica)
- se $x \sim y$ e $y \sim z$ então $x \sim z$ (transitiva)

A primeira propriedade é denominada *reflexiva* e informa que se uma relação é de equivalência então um elemento pode ser comparado a si mesmo. A segunda propriedade chamada *simétrica* nos informa que se um elemento x pode ser comparado com um elemento y então o elemento y também pode ser comparado com o elemento x e finalmente a terceira propriedade chamada *transitiva* nos informa que se um elemento pode ser comparado a um segundo elemento e este por sua vez pode ser comparado a um terceiro elemento, então o primeiro elemento pode ser comparado com o terceiro elemento.

Após esses conceitos iniciais, surge o questionamento sobre qual a utilidade de aplicar o conceito de *relação de equivalência* num conjunto e a resposta é simples:

relações de equivalência são usadas para classificar os elementos de um conjunto em subconjuntos com propriedades semelhantes, sendo cada subconjunto produzido por essa classificação chamado de *classe de equivalência* [1].

Por tratar-se de um tema matemático, é conveniente uma maior formalidade, portanto segue-se a notação usada em [1]: seja **X** um conjunto e $\sim$ um relação definida em **X** , se $x \in \mathbf{X}$ então a *classe de equivalência* de *x* é o conjunto dos elementos de **X** que são equivalentes a *x* por $\sim$ e matematicamente denotada por:

$$\bar{x} = \{ y \in \mathbf{X} : y \sim x\}$$

Como dito acima uma *relação de equivalência* divide um conjunto em *classes de equivalência* e tal divisão possui a seguinte propriedade: *qualquer elemento de uma classe de equivalência é um representante de toda a classe*, isto é, se um elemento da classe é conhecido, conhece-se toda a classe. Tal propriedade é matematicamente simbolizada por: se $x \in \mathbf{X}$ e $y \in \bar{x}$ então $\bar{x} = \bar{y}$. Os interessados na prova dessa propriedade podem consultar [1].

E para finalizar esta pequena introdução ao estudo de *relações de equivalência* enuncia-se as seguintes propriedades do conjunto **X** com a *relação de equivalência* $\sim$:

- a união de todas as *classes de equivalência* é o conjunto **X**;
- a intersecção de duas *classes de equivalência* é disjunta.

2.7.2 Inteiros Módulo *n*

No conjunto dos números inteiros pode-se definir a seguinte relação de congruência: dado um número *n* qualquer, dois inteiros *x* e *y* são equivalentes se a diferença entre eles é um múltiplo de *n*. Formalmente, diz-se que *x* e *y* são *congruentes módulo n* se x - y é um múltiplo de *n* e escreve-se simbolicamente como:

$$x \equiv y \pmod{n}$$

A prova que essa relação de congruência constitui-se uma relação de equivalência pode ser encontrada em [1].

Em nosso estudo, o conjunto que de fato interessa é o conjunto quociente de $\mathbb{Z}$ pela relação de congruência módulo *n*, conjunto este denominado *conjunto dos inteiros módulo n* e representado por $\mathbb{Z}_n$. Dado que essa relação de congruência divide o *conjunto dos inteiros módulo n* em classes de equivalência, faz-se necessário identificar os elementos dessas classes e para tal considere a notação usada

em [1]: seja $a \in \mathbb{Z}$, a classe *a* é formada pelos $b \in \mathbb{Z}$ que satisfazem *b - a* é múltiplo de *n*, isto é, *b - a = K n*, para algum $K \in \mathbb{Z}$. Simbolicamente pode-se representar a classe *a* por: $\bar{a} = \{ a + Kn : K \in \mathbb{Z} \}$.

Dessa forma pode-se notar que dado um inteiro *n* e a relação de congruência acima denotada, particiona-se o conjunto dos inteiros em *n* classes de equivalência, em outras palavras, o conjunto quociente $\mathbb{Z}_n$ é formado pelas classes $\bar{0}, \bar{1} \cdots \overline{n-1}$. É interessante notar também que a única maneira de dois números entre *0* e *n - 1* serem congruentes módulo *n* é se forem iguais e assim pode-se representar $\mathbb{Z}_n$ por: $\mathbb{Z}_n = \{ \bar{0}, \bar{1} \cdots \overline{n-1} \}$. Finalmente cumpre ressaltar a possibilidade de representar cada classe de $\mathbb{Z}_n$ na forma reduzida seguindo a notação apresentada em [1], assim quando uma classe estiver representada na forma $\bar{a}$ com $0 \leq a \leq n - 1$, diz-se que está na *forma reduzida*.

2.7.3 Aritmética Modular

Como visto na seção anterior pode-se particionar o conjunto dos inteiros em *n* classes disjuntas através da *relação de congruência módulo n* e fez-se tal partição considerando-se o resto da divisão de um inteiro por *n*.

Passados todos esses conceitos iniciais, pode-se pensar numa aritmética aplicável ao conjunto $\mathbb{Z}_n$ e este é o propósito desta seção. Salienta-se que serão apenas apresentadas as operações definidas em $\mathbb{Z}_n$ e suas respectivas propriedades, sem no entanto provar tais propriedades. Ao leitor interessado no assunto sugere-se consultar [1].

Dessa forma apresenta-se as seguintes operações em $\mathbb{Z}_n$:

- adição,
- subtração,
- multiplicação,
- divisão.

A soma em $\mathbb{Z}_n$ é definida usando a operação soma de inteiros e é representada da seguinte forma: $\bar{a} + \bar{b} = \overline{a+b}$. Para verificar tal igualdade considere a soma $\bar{5} + \bar{5}$ no conjunto $\mathbb{Z}_8$, logo pela definição de soma tem-se que $\bar{5} + \bar{5} = \overline{5+5}$ e portanto $\bar{5} + \bar{5} \equiv 2 \pmod 8$, pois $\bar{10} \equiv 2 \pmod 8$.

A subtração é definida de maneira análoga não oferecendo maiores dificuldades. Apenas para exemplificar considere a seguinte subtração em $\mathbb{Z}_8$: $\bar{13} - \bar{5}$ que terá como resultado $\bar{0}$, pois $\bar{13} - \bar{5} = \overline{13-5}$ e $\bar{8} \equiv 0 \pmod 8$.

A multiplicação é definida da seguinte maneira: $\bar{a} \times \bar{b} = \overline{ab}$ e por ser semelhante a adição apresentada julga-se não necessária a apresentação de um exemplo.

Para definir a divisão em $\mathbb{Z}_n$ considere *x* e *y* dois números quaisquer e com $y \neq 0$ e interpreta-se a divisão de *x* por *y* como a seguinte multiplicação $x \times \frac{1}{y}$, sendo o número $\frac{1}{y}$ conhecido como o *inverso* de *y*. Transpondo tal definição para $\mathbb{Z}_n$ e supondo que deseja-se dividir $\bar{2}$ por $\bar{3}$ em $\mathbb{Z}_8$, pode-se transformar tal divisão numa multiplicação como indicado acima e posteriormente resolver tal multiplicação. Entretanto ao aplicar-se tal método depara-se com o problema de encontrar o inverso de um número em $\mathbb{Z}_n$ e para atacar tal problema pode-se recorrer ao seguinte teorema apresentado em [1]:

Teorema 2.7.1 (Teorema de inversão) *A classe $\bar{a}$ tem inverso em $\mathbb{Z}_n$ se, e somente se, a e n são primos entre si.*

O teorema acima nos diz que somente é possível dividir em $\mathbb{Z}_n$ se *n* e o divisor forem primos entre si, portanto voltando ao exemplo dado acima verifica-se que o *mdc(3,8)= 1*, ou seja, são primos entre si e portanto a divisão do exemplo está definida e para realizá-la basta encontrar-se o inverso de $\bar{3}$ em $\mathbb{Z}_8$ e multiplicar-se por $\bar{2}$, dado que este era o dividendo no exemplo considerado. Assim sendo, a última questão a solucionar é como encontrar o inverso de uma classe em $\mathbb{Z}_n$ e para tal pode-se utilizar o algoritmo euclidiano estendido .

Definidas essas quatro operações em $\mathbb{Z}_n$ resta-nos enumerar algumas propriedades dessas operações, sem no entanto apresentar provas de tais propriedades.

A adição em $\mathbb{Z}_n$ goza das seguintes propriedades:

- associativa: $(\bar{a} + \bar{b}) + \bar{c} = \bar{a} + (\bar{b} + \bar{c})$;
- comutativa: $\bar{a} + \bar{b} = \bar{b} + \bar{a}$;
- elemento neutro: $\bar{a} + \bar{0} = \bar{a}$;
- simétrica: $\bar{a} + \overline{-a} = \bar{0}$.

A multiplicação em $\mathbb{Z}_n$ goza das seguintes propriedades:

- associativa: $(\bar{a} \times \bar{b}) \times \bar{c} = \bar{a} \times (\bar{b} \times \bar{c})$;
- comutativa: $\bar{a} \times \bar{b} = \bar{b} \times \bar{a}$;
- elemento neutro: $\bar{a} \times \bar{1} = \bar{a}$.

E além dessas propriedades há uma propriedade em comum entre a multiplicação e a adição, tal propriedade é a *distributiva*:

$$\bar{a} \times (\bar{b} + \bar{c}) = \bar{a} \times \bar{b} + \bar{a} \times \bar{c}$$

2.7.4 A Função de Euler

Considerando um inteiro positivo *n* e denotando-se por *U(n)* o conjunto dos elementos inversíveis em $\mathbb{Z}_n$, pode-se associar a esse conjunto uma função que indique a ordem do mesmo, ou seja, quantos elementos o conjunto *U(n)* possui e essa função é a *função ϕ de Euler* ou *função totiente* representada simbolicamente por $\phi(n)$.

Definida a *função totiente* necessita-se de um método para calcular o seu valor. Para tal considere um caso particular em que seja dado um número *p* primo e por conseguinte todos os inteiros positivos menores que *p* são primos com *p*, logo $\phi(p)$= p - 1. Para calcular o valor de $\phi(n)$ em casos gerais faz-se uso do seguinte teorema:

Teorema 2.7.2 (Teorema) *Se m,n são inteiros positivos tais que mdc(m,n)=1, então* $\phi(mn) = \phi(m)\phi(n)$.

A prova do teorema acima e da fórmula geral para $\phi(n)$ pode-se encontrar em [1].

2.7.5 O Teorema de Fermat

Este teorema somente será enunciado, deixando para o leitor interessado no assunto consultar em [1] uma prova do mesmo.

Teorema 2.7.3 (Teorema de Fermat) *Seja p um número primo e a um número inteiro, então* $a^p \equiv a \pmod{p}$.

2.7.6 O Teorema de Euler

O seguinte teorema creditado a Euler será aqui apenas enunciado, visto sua necessidade para provar algumas propriedades do algoritmo de criptografia RSA. Uma prova deste teorema pode ser encontrada em [1].

Teorema 2.7.4 (Teorema de Euler) *Sejam* $n \succ 0$ *e a números inteiros. Se* mdc(a,n)= 1 *então* $a^{\phi(n)} \equiv 1 \pmod{n}$.

2.7.7 O Algoritmo Estendido de Euclides

O algoritmo de Euclides tem por objetivo calcular o máximo divisor comum entre dois números inteiros, entretanto pode-se estender *o algoritmo de Euclides* para achar-se o máximo divisor comum entre dois inteiros e encontrar-se também dois inteiros α e β tais que: $\alpha \times a + \beta \times b = d$, onde *a* e *b* são números inteiros positivos e *d* é o máximo divisor comum entre eles.

O algoritmo de Euclides assim modificado é conhecido como *algoritmo euclidiano estendido* e [1] apresenta uma versão desse algoritmo proposta por Knuth.

2.8 O Método de Criptografia RSA

2.8.1 Descrição do Algoritmo RSA

Este método de criptografia com chave pública ou assimétrico foi proposto em 1977 por Rivest, Shamir e Adleman e seu nome deriva das iniciais dos nomes de seus inventores. O uso de chave pública é baseado no trabalho de Diffie e Hellman [6] e visa resolver um dos grandes problemas dos métodos criptográficos simétricos, a saber: a distribuição das chaves. Tal problema consiste na necessidade de enviar a chave de criptografia por um canal inseguro, o que exige um protocolo complexo e uma grande quantidade de troca de dados para tentar evitar que algum usuário não legítimo descubra a chave de criptografia. A criptografia assimétrica elimina este problema porque a chave que deveria ser enviada pelo canal inseguro é de antemão conhecida, pois é uma chave pública.

A primeira etapa necessária ao uso do RSA é converter o texto simples numa seqüência de números, etapa essa chamada de pré-codificação [1]. Nesta etapa faz-se com que cada caracter do texto a ser codificado corresponda a um código numérico, sendo que tal codificação pode ser feita utilizando-se uma tabela de conversão própria ou o código ASCII do caracter. Entretanto deve-se garantir que o código de cada caráter seja do mesmo tamanho afim de evitar ambigüidades, pois se código numérico para a letra A for o número 1 e o da letra B for o número 2 e assim sucessivamente, como interpretar o código numérico 12? Ele pode ser tanto a seqüência AB como a letra L e não seria possível fazer a escolha correta.

Passada a fase de conversão do texto simples em código numérico, precisa-se definir os parâmetros a serem usados no RSA; esses parâmetros são dois números primos distintos *p* e *q* e o número *n* produto de tais primos. Definidos esses números executa-se o próximo passo da pré-codificação: dividir em blocos o longo número formado pela conversão citada no parágrafo anterior, sendo que tais blo-

cos devem ter tamanhos menores que *n*. Cabe salientar que essa divisão em blocos não é única, devendo-se entretanto evitar que um bloco comece com zero, pois isso faria com que perdesse informações quando da decodificação [1] e na prática o tamanho do bloco usado é o maior valor menor que *n*[6].

É interessante perceber que a divisão em blocos traz uma vantagem adicional ao método RSA, pois tais blocos não correspondem a nenhuma unidade lingüística, o que torna a decodificação por contagem de freqüência essencialmente impossível [1].

Finalizada a fase de pré-codificação, passa-se à fase de codificação e decodificação. Para codificar um texto simples usando o método de criptografia RSA precisa-se de dois parâmetros: o número *n* que é o produto dos dois números primos *p* e *q* e de um inteiro positivo *e* que seja inversível módulo $\phi(n)$ (vide seção 2.7.4). Em outras palavras tem-se que mdc(e,$\phi(n)$)=1. A chave pública de codificação do RSA será o par (n,e). Assim, dado um texto que foi submetido ao processo de pré-codificação, tem-se uma seqüência de blocos numéricos como resultado, seqüência esta que será codificada usando-se cada bloco separadamente, os quais devem manter-se separados após a codificação, caso contrário não será possível decodificar o texto.

Para tornar mais claro o processo de codificação considere o exemplo citado em [1]. Suponha que a chave de codificação é *(n,e)* e deseja-se codificar um bloco *b*, o qual é um inteiro menor que *n*. Denota-se por **C** (*b*) o bloco *b* codificado e dessa forma **C** (*b*) $\equiv b^e \pmod{n}$. Dessa forma ao codificar-se um bloco *b*=102, com os valores de *p* =11, *q* =13, *n*=143 e $\phi(n)$=120 e *e* = 7, tem-se que: $102^7 \equiv 119 \pmod{143}$. e logo o bloco *b* =102 é codificado usando o método RSA com o valor 119.

Para decodificar um bloco precisa-se da chave privada de decodificação composta do par *(n,d)*, onde o valor de *d* é o inverso de *e* em $\phi(n)$ e é calculado pelo algoritmo euclidiano estendido aplicado a *e* e $\phi(n)$ (vide seção 2.7.4). Dessa forma dado um bloco *b* de texto codificado, decodifica-o aplicando a seguinte fórmula: **D** (*b*) $\equiv b^d \pmod{n}$. E assim decodifica-se o exemplo dado no parágrafo anterior: $119^{103} \equiv 102 \pmod{143}$, ou seja, o valor 119 decodificado pelo RSA produz como resultado 102 e portanto **D(C(b))=b**. Este resultado é o esperado, visto que se decodificando um bloco do texto codificado não encontrar-se o bloco do texto simples correspondente tem-se um código inútil.

Vê-se nos exemplos dos dois parágrafos anteriores que **D(C(b))=b**, mas no entanto não apresenta-se nenhuma prova de que este fato é verdadeiro em quaisquer circunstâncias por isso sugere-se ao leitor interessado nessa prova consultar [1],

onde é dada um prova de tal fato utilizando-se dos teoremas de Euler (vide seção 2.7.6) e Fermat (vide seção 2.7.5).

2.8.2 Segurança do Algoritmo RSA

A segurança do RSA advém da dificuldade de se fatorar números grandes [6], sendo este um problema que parece ser extremamente difícil visto que matemáticos vêm tentando fatorar números extensos há pelo menos 300 anos e os resultados obtidos ainda não são satisfatórios [9], ou seja, não se conhece nenhum algoritmo eficiente para a resolução de tal problema. Em [6] encontra-se que o tempo do melhor algoritmo de fatoração é da ordem de $O(e^{sqrt(lnn \times ln(lnn))})$ e com este tempo um computador que realize um milhão de passos por segundo levaria $3,8 \times 10^9$ anos para fatorar um número de 200 dígitos.

Para entender porque a segurança do RSA está baseada na dificuldade de fatorar números extensos considere as afirmações feitas em [1]: o RSA é um método de chave pública e portanto o par(n,e) é acessível a qualquer usuário e a segurança deste método é assegurada se for difícil calcular *d* quando apenas o par *n* e *e* são conhecidos. Dado este fato pode-se verificar que realmente é difícil tal cálculo de *d*, pois sabe-se calcular seu valor somente aplicando o algoritmo euclidiano estendido a $\phi(n)$ e *e* e para encontrar $\phi(n)$ precisa-se fatorar *n* e obter os parâmetros *p* e *q*.

Neste ponto pode-se indagar a possibilidade de encontrar $\phi(n)$ sem fatorar *n*, entretanto conhecendo os dois valores pode-se chegar aos fatores de *n* e portanto fatora-se o mesmo; aos interessados numa prova deste fato sugere-se consultar [1].

Outro fator relevante para a segurança do RSA é a escolha dos parâmetro *p* e *q*, visto que se forem pequenos é evidente a facilidade para se quebrar o código e mesmo para valores grandes deve-se tomar o cuidado para que a diferença entre os dois não seja pequena, pois se isso acontecer pode-se encontrar *p* e *q* facilmente usando o algoritmo de Fermat [1].

Um exemplo da facilidade de quebrar o sistema RSA quando os parâmetros *p* e *q* não são cuidadosamente escolhidos aconteceu com o *Netscape* em 1995, quando dois estudantes da Universidade da Califórnia descobriram que era possível decifrar qualquer mensagem da versão do RSA usada no *Netscape* em poucas horas.

Deve-se também considerar a segurança do RSA em função do tamanho das chaves usadas, visto que com o aumento do poder computacional torna-se mais fácil a decifragem de mensagens que são codificadas com chaves de poucos bits. Para dar uma idéia da evolução da criptoanálise do RSA em função do tamanho das

chaves reproduz-se a figura 2.2 com o histórico de quebras das chaves do método de criptografia RSA. Tal figura encontra-se em [3]

Dígitos Decimais	Bits(aprox.)	Data da Quebra
100	332	1991
110	365	1992
120	398	1993
129	428	1994
130	431	1996
154	512	1999

Figura 2.2: Histórico de quebra do RSA

Como pode ser visto em [3], atualmente sugere-se o uso de chaves de 1024 bits (319 algarismos) para informações de uso pessoal e 2048 bits (617 algarismos) para uso militar e industrial.

E para finalizar essa breve discussão sobre a segurança do método RSA pode-se citar dois fatos descobertos recentemente. Em 1995 um consultor em segurança de computadores mostrou que o uso pouco cuidadoso da técnica de autenticação de assinaturas torna vulneráveis métodos de chaves públicas como o RSA; a técnica consiste em enviar uma mensagem assinada e marcar o tempo que o sistema leva para confirmar a assinatura e repetindo o processo com mensagens de tamanhos ligeiramente diferentes é possível obter informações suficientes para descobrir a chave de decodificação que está sendo usada. E a publicação do algoritmo de Shor em 1994 mostrou que se um computador quântico puder ser construído tem-se uma maneira de fatorar números grandes rapidamente e isto tornaria o RSA totalmente obsoleto[1].

2.9 O Algoritmo de Criptografia Posicional

O Algoritmo de criptografia Posicional desenvolvido em 2001 por Moreno e Chiaramonte [2] propõe que a posição do caracter no texto simples interfira na chave de criptografia utilizada. Para isto introduziu-se uma função polinomial $f(x)$ sobre o algoritmo da cifra de César (vide seção 2.2.1). Assim ao codificar um símbolo considera-se o valor numérico da tabela ASCII para este símbolo, a esse valor soma-se a imagem da função polinomial $f(x)$, onde x é a posição do símbolo no texto simples e finalmente aplica-se a operação $(\mathrm{mod}\ 256)$ sobre resultado da soma. Matematicamente a função de criptografia é a seguinte:

$$ValorCodificado = (ASCII + f(x))\ (\mathrm{mod}\ 256)\ 2.1$$

sendo que o uso da operação *mod* garante que o tamanho do texto codificado seja o mesmo do texto simples.

A função polinomial $f(x)$ pode ser de qualquer grau e com coeficientes inteiros, os quais constituem a chave de codificação. Conforme [2] a chave de codificação pode ser selecionada aleatoriamente. Em [2] vê-se que quanto maior o grau da função polinomial, maior a robutez do sistema criptográfico.

Para uma melhor compreensão do processo de codificação do algoritmo posicional considere a tabela 2.1 onde será codificada a palavra MARIO usando como chave os números *6, 13 e 5* que são os coeficientes da função polinomial $f(x) = 6x^2 + 13x + 5$.

Seqüência	M	A	R	I	O
Valor na tabela ASCII	77	65	82	73	79
Valor da posição	1	2	3	4	5
Resultado da expressão	24	55	98	153	220
Resultado + ASCII	101	120	180	226	299
Valor criptografado	101	120	180	226	43

Tabela 2.1: Exemplo de codificação usando o Algoritmo Posicional

Para decodificar usa-se a seguinte expressão matemática:

$$(c - f(x))\ (\mathrm{mod}\ 256)\ 2.2$$

onde: c é o valor codificado do símbolo na posição x e $f(x)$ é a imagem da chave de criptografia também para o símbolo da posição x. Portanto usa-se a mesma chave para decodificar os dados e assim tem-se um método de criptografia simétrico.

Na tabela 2.2 vemos a decodificação da mensagem mostrada na tabela 2.1 .

Valor criptografado	101	120	180	226	43
Valor da posição	1	2	3	4	5
Valor da expressão	24	55	98	153	220
Valor decodificado	77	65	82	73	79
Seqüência	M	A	R	I	O

Tabela 2.2: Exemplo de decodificação usando o Algoritmo Posicional

2.10 Noções de Teoria da Complexidade de Algoritmos

A teoria da complexidade de algoritmos disponibiliza métodos que possibilitam o estudo do comportamento do algoritmos sem que haja dependência do equipamento.

Segundo [7] para medir o custo de execução de um algoritmo define-se uma função custo ou função de complexidade *f*, onde $f(n)$ é a medida de tempo necessário para executar um algoritmo considerando-se um problema de tamanho *n*. Ressalta-se que a complexidade de tempo não representa tempo diretamente, mas o número de vezes que determinada expressão ou operação considerada relevante é executada.

Pode-se distinguir três casos:

- **melhor caso:** corresponde ao menor tempo de execução sobre todas as entrada possíveis de tamanho *n*;
- **caso médio:** corresponde à média dos tempos de execução de todas as entradas de tamanho *n*;
- **pior caso:** corresponde ao maior tempo de execução sobre todas as entradas possíveis de tamanho *n*

Como visto em [7] se *f* é uma função de complexidade baseada na análise do pior caso, então o custo de usar o algoritmo nunca é maior que $f(n)$.

A análise de algoritmos preocupa-se com os grandes valores de *n*, isto é, estuda-se o comportamento assintótico da função de complexidade $f(n)$, entendendo-se por comportamento assintótico o limite da função $f(n)$ quando *n* tende ao infinito.

Uma notação para descrever a dominação assintótica foi proposta por Knuth, conforme visto em [7] e denomina-se tal notação de **Notação O**, matematicamente definida como:
Uma função $g(n)$ é **O(** $f(n)$ **)** se existem duas constantes positivas *c* e *m* tais que $g(n) \leq c \times f(n)$, para $n \geq m$.

Assim afirmar que $g(n)$ é **O(** $f(n)$ **)** significa que $f(n)$ é o limite superior para a taxa de crescimento de $g(n)$ [7].

Ao analisar um algoritmo pode-se seguir um conjunto de regras e entre elas cita-se: o tempo de execução de um comando de atribuição, leitura ou escrita é considerado como **O(1)** [7] e o tempo de execução de operações de ULA (unidade lógica aritmética), tais como soma, multiplicação ou exponenciação são constantes por tipo de parâmetro [4]. O tipo de parâmetro pode ser um: inteiro, double ou float.

Capítulo 3

Avaliação da Eficiência e Segurança do Algoritmo Posicional

3.1 Introdução

Apresenta-se neste capítulo a maneira como verificou-se a eficiência do algoritmo posicional e como analisou-se a segurança do mesmo.

3.1.1 Analisando a Eficiência do Algoritmo Posicional

Neste trabalho analisou-se a eficiência do algoritmo de criptografia Posicional considerando-se como parâmetro para tal medida a complexidade computacional e o tempo medido na prática para criptografar arquivos de diferentes tamanhos. Com esta análise procurou-se determinar se a complexidade do algoritmo em função do tamanho do arquivo é linear ou exponencial.

Análise da Complexidade do Algoritmo Posicional

Para a avaliação da eficiência implementou-se o algoritmo posicional na linguagem de programação C++ usando-se a biblioteca *gmp.h* na versão 3.1.1 para trabalhar-se com números grandes e compilou-se usando o compilador *g++* na versão 2.96. Avaliou-se o tempo gasto para criptografar arquivos dos seguintes tamanhos: 0.5 MBytes, 1.0MBytes, 1.5MBytes e assim sucessivamente até 10.0MBytes. No ar-

tigo publicado pelos autores do algoritmo posicional testou-se arquivos de tamanho igual a 0.5MBytes,1.0MBytes e 1.5 MBytes apenas.

Implementou-se também o algoritmo de criptografia RSA na linguagem C++ e usando-se a biblioteca *gmp.h* na versão 3.1.1 para trabalhar-se com números grandes e compilou-se usando o compilador *g++* na versão 2.96. Executou-se tal algoritmo com chave de 16 bits e mediu-se o tempo gasto pelo mesmo para criptografar arquivos de tamanhos: 0.5 MBytes, 1.0 MBytes, 1.5MBytes e assim sucessivamente até 10.0 MBytes. As medidas obtidas permitiram fazer uma comparação entre o algoritmo posicional e o RSA. Em [2] também utiliza-se o RSA como parâmetro comparativo.

Embora o algoritmo de criptografia posicional seja um método simétrico e o algoritmo RSA seja assimétrico, ressalta-se que a comparação com o algoritmo RSA se justifica pelo fato de ambos os métodos utilizarem operações matemáticas semelhantes (multiplicação, exponenciação e módulo) durante o processo de codificação.

Para analisar a complexidade computacional do algoritmo posicional considerou-se o seguinte pseudocódigo do programa implementado neste trabalho:

Início
1: n=número de caracteres do arquivo
2: enquanto n≥ 1 faça
3: leia caracter
4: (caracter lido + $f(x)) \mod 256$
Fim

Este algoritmo apresenta um comportamento linear em função do tamanho do arquivo, pois a operação de leitura e as operações $(caracterlido + f(x)) \mod 256$ são executadas num tempo constante (vide seção 2.10). Dessa forma a função que mede a eficiência do algoritmo é $f(n) = c \times n$, onde *c* é uma constante que representa a soma do tempo gasto para executar a leitura do caracter e as operações de ULA necessárias à codificação do mesmo, e *n* é o número de caracteres a ser criptografado. Logo $f(n) = O(n)$ e tal fato verificou-se na prática através dos testes realizados, conforme será visto na próxima seção.

Análise Prática do Algoritmo Posicional

Realizou-se os testes num computador Pentium III 550 MHZ, com 128 MBytes de memória RAM e rodando a versão 2.4.3-12 do sistema operacional Linux. Utilizou-se o comando *time* do *shell tcsh* para computar o tempo gasto pelo al-

goritmo ao criptografar um arquivo. O comando *time* retorna o tempo gasto executando rotinas do usuário e o tempo gasto executando rotinas do sistema, portanto para encontrar o tempo total gasto na execução do algoritmo, somou-se o tempo gasto pelo sistema ao tempo gasto executando rotinas do usuário.

As tabelas 3.1 e 3.2 mostram na primeira linha o tamanho dos arquivos e na primeira coluna o grau da função de criptografia, tendo como intersecção o tempo (em segundos) gasto para codificar o arquivo. Nestes testes todos os coeficientes da chave de criptografia são iguais a um.

	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
1	1,83	3,68	5,54	7,37	9,24	11,37	12,94	14,77	16,47	18,51
2	2,25	4,53	6,83	9,13	11,44	14,06	16,05	18,35	20,31	22,95
3	2,71	5,41	8,13	10,94	13,61	17	19,56	23,02	24,39	27
4	3,51	7,1	10,72	14,3	17,9	22,06	25,15	28,78	31,72	36,02
5	4,09	8,29	12,99	16,72	20,97	25,86	29,44	33,82	36,91	42,31
6	4,75	9,63	14,53	19,4	24,34	30,06	34,31	39,43	42,89	49,4 9
7	5,44	10,94	16,93	22,14	27,73	34,35	39,34	45,07	49,84	57,05
8	6,18	12,54	18,82	25,2	31,59	39,14	44,74	51,34	57,51	64,43
9	7,03	14,32	21,12	28,91	36,18	44,99	51,62	59,53	67,03	74,73
10	7,86	16,23	25,01	33,04	41,47	51,27	58,76	67,38	73,42	85,16

Tabela 3.1: Tempo de Criptografia do Algoritmo Posicional

	5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10.00
1	20,13	21,96	23,79	25,62	27,33	29,28	31,11	32,94	34,77	36,9
2	24,5	27	29,25	31,5	34	36,3	38,36	42,86	42,64	45,96
3	29,81	32,52	34,77	37,02	40,73	43,06	46,27	48,89	53,01	58,82
4	38,61	42,12	45,63	49,14	53,15	56,16	59,67	63,18	66,69	72,24
5	45,06	49,24	53,17	57,67	63,17	66,45	69,55	73,62	78,63	85,28
6	54,25	58,02	63,24	68,45	73,69	78,62	81,64	86,7	91,3	100,09
7	62,14	67,28	71,73	78,19	84,44	89,04	94,82	99,72	106,87	114,04
8	71,72	78,24	84,76	90,03	96,07	102,12	110,98	116,84	122,98	130,46
9	80,56	87,71	95,02	102,9	112,11	119,97	125,46	133,02	143,17	153,27
10	88,61	96,72	104,97	116,1	126,55	129,44	137,87	154,31	167,35	170,72

Tabela 3.2: Tempo de Criptografia do Algoritmo Posicional

A figura 3.1 mostra o gráfico correspondente às tabelas 3.1 e 3.2.

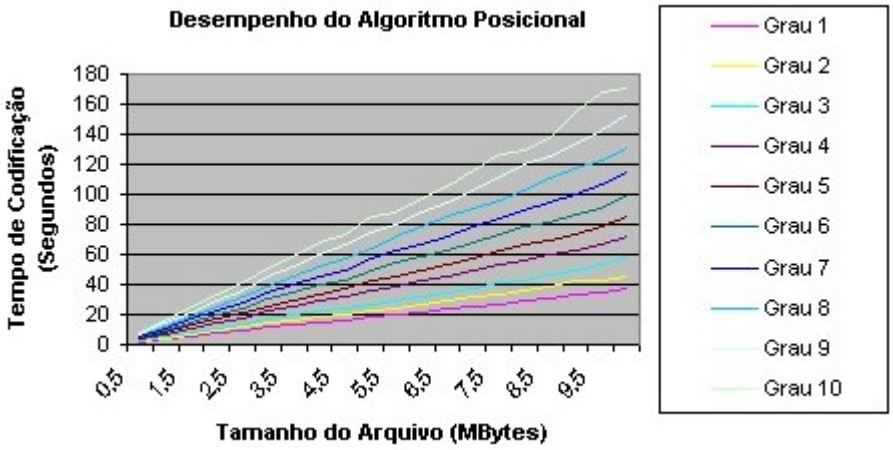

Figura 3.1: Tempo de Criptografia do Algoritmo Posicional

Realizou-se também testes para verificar a influência dos coeficientes da função de criptografia, mantendo-se o grau constante, no tempo de codificação de um arquivo. As tabelas 3.3, 3.4 e 3.5 mostram, respectivamente, os resultados obtidos usando funções de grau três, seis e 10. Nessas tabelas a primeira linha mostra o tamanho do arquivo (em MBytes) e a primeira coluna os coeficientes da função de criptografia, tendo na intersecção o tempo (em segundos) gasto na codificação do arquivo.

	1.0	10.0
1	5,41	58,82
127	5,42	59,36
255	5,39	59,79

Tabela 3.3: Tempo de Criptografia do Algoritmo Posicional com função de grau 3

	1.0	10.0
1	9,63	100,09
127	9,65	100,09
255	9,61	100,31

Tabela 3.4: Tempo de Criptografia do Algoritmo Posicional com função de grau 6

	1.0	10.0
1	16,23	170,7
127	16,09	171,6
255	16,46	172,3

Tabela 3.5: Tempo de Criptografia do Algoritmo Posicional com função de grau 10

A figura 3.2 apresenta um gráfico em 3D, referente as tabelas 3.1 e 3.2, mostrando-se a variação do tempo (em segundos) gasto para codificar um arquivo em função do seu tamanho (em MB) e do grau da função de criptografia. Percebe-se pelo gráfico que quanto maior o arquivo e o grau da função de criptografia, maior o tempo gasto na codificação.

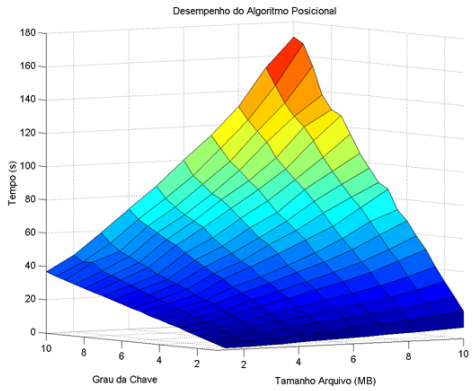

Figura 3.2: Tempo de Criptografia do Algoritmo Posicional

Análise Prática do Algoritmo RSA

Mediu-se também o tempo gasto pelo RSA para codificar arquivos com tamanho variando de 0.5 MBytes até 10.0 Mbytes e chave de 16 bits. Tais testes foram

realizados num computador equivalente ao usado para os testes com o algoritmo posicional e obteve-se os resultados apresentados nas tabelas 3.6 e 3.7.

tamanho(MB)	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
tempo(s)	34,5	68	101,51	139,8	170,36	208,98	238,26	272,73	310,5	340,51

Tabela 3.6: Tempo de Criptografia do Algoritmo RSA

tamanho(MB)	5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10.00
tempo(s)	380,6	416,14	449,8	485,4	512,07	553,2	587,88	621,72	654,21	687,69

Tabela 3.7: Tempo de Criptografia do Algoritmo RSA

A figura 3.3 mostra o gráfico correspondente às tabelas 3.6 e 3.7.

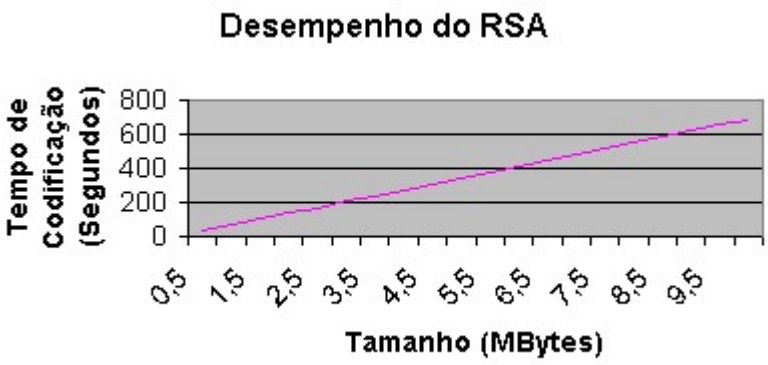

Figura 3.3: Tempo de Criptografia do Algoritmo RSA

3.2 Analisando a Segurança do Algoritmo Posicional

Ao analisar-se a segurança do algoritmo posicional verificou-se a influência da operação matemática *mod* na construção das chaves de codificação e a possibilidade de quebrar o algoritmo usando-se os métodos da força bruta e de análise de freqüência.

A operação matemática *mod* induz uma partição no conjunto dos números inteiros como visto na seção 2.7.2. Como visto na seção 2.9, o algoritmo posicional

utiliza a operação matemática $mod256$ na chave de codificação, a qual é formada por números inteiros. Portanto esta operação divide o universo de chaves possíveis em *256* classes de equivalência e assim sendo cada coeficiente da função $f(x)$ usada na codificação pode variar de *0* a *255*.

Como visto acima temos *256* possibilidade diferentes para cada coeficiente inteiro da função $f(x)$ usada na codificação e isto reduz o universo de chaves possíveis, visto que se não utilizasse a operação matemática $mod256$ na chave de criptografia teria-se infinitas possibilidades para cada coeficiente. A infinidade de chaves possíveis ao excluir-se a operação $mod256$ deve-se ao fato de cada coeficiente da função $f(x)$ é inteiro e o conjunto dos números inteiros possui cardinalidade infinita.

Para calcular o número de chaves possíveis em função do grau de $f(x)$ usou-se a seguinte fórmula: 256^{n+1}, onde *n* é o grau de $f(x)$. Essa fórmula é deduzida considerando todas as combinações possíveis entre os coeficientes de $f(x)$, como para cada coeficiente existem *256* possibilidades diferentes e o número de coeficientes numa função $f(x)$ de grau *n* é *n + 1*, tem-se então que o número de chaves possíveis é 256^{n+1}.

Percebe-se que mesmo a operação $mod256$ reduzindo o universo de chave possíveis, tem-se que a quantidade de chaves possíveis para um teste usando a força bruta aumenta exponencialmente em função do grau da função de codificação.

O método da análise de freqüência consiste em analisar a freqüência dos símbolos num determinado idioma e fazer-se o mesmo no texto codificado. A partir dessa análise supor que o caráter de maior freqüência num idioma coincide com o caráter de maior freqüência no texto codificado e com essa informação tentar inferir a chave de codificação usada no processo.

Dessa maneira para executar tal análise implementou-se dois programas: um programa para analisar a freqüência dos símbolos num idioma qualquer e outro para analisar a freqüência dos símbolos no texto criptografado.

Visto que na função de criptografia do algoritmo posicional aplica-se a operação matemática $mod256$, ao fazer-se a análise de freqüência no texto criptografado considerou-se os caracteres nas posições múltiplas de 256 (assim começando da primeira posição o próximo símbolo a ser verificado é o da posição 257, começando da posição 2 o próximo símbolo a ser analisado é o da posição 258 e assim sucessivamente).

Procedeu-se da maneira acima visto que apenas caracteres nas posições múltiplas de *256* são codificados da mesma forma e pode-se demonstrar tal fato da seguinte maneira: cada caracter é criptografado com a expressão $(f(x) + ASCII)$

$(\mathrm{mod}\ 256)$, onde *x* é a posição do caracter no texto simples; dessa forma o valor de $f(x)$ repete-se a cada intervalo de *256* posições e sendo o valor de $f(x)$ o mesmo para dois caracteres iguais, esse caracteres serão codificados com os mesmos valores. Considere $f(x) = x^2 + x + 1$ e ASCII=65, então:
$(f(1) + 65)\ (\mathrm{mod}\ 256) = (3 + 65)\ (\mathrm{mod}\ 256) = 68$
$(f(2) + 65)\ (\mathrm{mod}\ 256) = (7 + 65)\ (\mathrm{mod}\ 256) = 72$
e assim para cada valor diferente de *x* tem-se um $f(x)$ diferente, a não ser quando *x* é múltiplo de 256, como mostrado abaixo:
$(f(257) + 65)\ (\mathrm{mod}\ 256) = (66307 + 65)\ (\mathrm{mod}\ 256) = 68$
$(f(258) + 65)\ (\mathrm{mod}\ 256) = (66823 + 65)\ (\mathrm{mod}\ 256) = 72$
assim $f(1) \equiv f(257)$ e $f(2) \equiv f(258)$ devido à operação $mod256$.

Capítulo 4

Discussão

4.1 Introdução

No capítulo 3 apresentou-se a avaliação do algoritmo de criptografia posicional quanto a eficiência e segurança. Neste capítulo apresenta-se uma discussão sobre os resultados obtidos nessas avaliações.

4.2 Eficiência do Algoritmo de Criptografia Posicional

4.2.1 Análise Teórica

O algoritmo de criptografia posicional apresenta um comportamento linear em função do tamanho do arquivo (vide seção 3.1.1) e comprovou-se tal fato na prática conforme apresentado na seção 3.1.1.

O comportamento linear do algoritmo posicional em função do tamanho do arquivo explica-se pelo fato do mesmo utilizar no processo de codificação operações matemáticas como a exponenciação, a multiplicação, adição e módulo (vide seção 2.9; e como visto na seção 2.10 tais operações apresentam um custo computacional constante. Assim sendo o custo computacional para codificar um arquivo com n caracteres é n vezes o tempo constante gasto na execução das operações aritméticas e dessa forma a complexidade do algoritmo é **O(n)**.

4.2.2 Análise Prática

Os testes realizados mostraram também que a eficiência do algoritmo posicional é dependente do grau da função de criptografia utilizada e do tamanho do

arquivo(vide figura 3.2).

Conforme visto na seção 3.1.1, nos estes práticos realizados com o algoritmo de criptografia posicional para verificar sua eficiência, utilizou-se o valor um como coeficiente da chave de criptografia. Utilizou-se tal valor porque ao realizar testes com valores maiores para a chave de criptografia obteve-se pequena variação no tempo gasto para codificar um arquivo (vide tabelas 3.3, 3.4 e 3.5). Pode-se interpretar este resultado como um indicativo que o grau da função de criptografia é um parâmetro de maior importância na medida da eficiência do algoritmo posicional que os valores das chaves utilizadas.

A dependência da eficiência do algoritmo posicional em função do grau da função de criptografia deve-se ao fato que quanto maior o grau maior o tamanho da imagem da função de criptografia e tal número pode ultrapassar o limite de bits usado para representá-lo no computador. Assim todas as operações aritméticas deve ser implementadas em *software*, fazendo uso de bibliotecas de grandes números ou múltipla precisão. Ao utilizar-se as bibliotecas de grandes números tem-se uma queda de desempenho, visto que tais bibliotecas necessitam alocar memória dinamicamente à medida que aumenta o tamanho dos números usados nas operações aritméticas realizadas. A queda na eficiência do algoritmo posicional ao aumentar o grau da função de criptografia pode ser verificado na figura 3.2, onde percebe-se claramente que ao aumentar o grau da função de criptografia e mantendo constante o tamanho do arquivo, tem-se um aumento no tempo de codificação.

A eficiência do algoritmo posicional também é dependente do tamanho do arquivo. Isto ocorre porque para grandes arquivos o valor da imagem de $f(x)$ pode ultrapassar o limite de bits usado para representá-lo no computador. Assim tem-se que usar bibliotecas de grandes números, as quais diminuem a eficiência do algoritmo pelo mesmo motivo visto no parágrafo anterior.

Ao comparar-se o algoritmo de criptografia posicional com o algoritmo RSA nota-se que o algoritmo posicional apresenta um melhor desempenho. Acredita-se que isso deve-se ao fato do algoritmo posicional trabalhar com números menores que o algoritmo RSA e dessa forma são feitos menos acessos a memória pela biblioteca utilizada para o tratamento de operações aritméticas com números grandes.

Finalmente cumpre ressaltar que o algoritmo posicional é simples de implementar e produz um texto codificado que é do mesmo tamanho do texto simples(vide seção 2.9).

4.3 Segurança do Algoritmo de Criptografia Posicional

Ao analisar a segurança do algoritmo de criptografia posicional verificou-se inicialmente a influência da operação matemática $mod256$ na chave de criptografia e conforme visto na seção 3.2 essa operação reduz o espaço de busca a ser executado pelo ataque da força bruta.

O número de chaves a serem testadas no ataque da força bruta é 256^{n+1}(vide seção 3.2), onde *n* é o grau da função de criptografia. Isso permite concluir que o grau da função de criptografia é um parâmetro para medir a segurança do algoritmo de criptografia posicional. Pode-se, por exemplo, verificar que o algoritmo de criptografia posicional usando uma função de criptografia de grau 10 tem 2^{88} chaves diferentes, pois, $256^{10+1} = (2^8)^{11} = 2^{88}$ e este valor é maior que o número de chaves a serem testadas no algoritmo de criptografia *DES*, o qual usa chaves de 56 bits e portanto existem 2^{56} chaves diferentes a serem testadas pelo método da força bruta (vide seção 2.3).

Considerando-se que atualmente sugere-se o uso de chaves de 2048 bits para o algoritmo RSA empregado nos meios militares e industriais, vê-se que para decifrar o código de tal algoritmo pelo método da força bruta seriam necessárias 2^{2048} tentativas no pior caso. Este número de tentativas equivale ao universo de chaves possíveis ao utilizar-se o algoritmo posicional com uma função de criptografia com grau 255. O tempo necessário ao executar as operações matemáticas necessárias ao funcionamento de tais algoritmos com números de tal ordem é grande forçando-se a construção de máquinas especiais que realizem tais cálculos em *hardware*.

O uso da operação matemática $mod256$ permite também concluir que não faz sentido usar como chave algum valor inteiro fora do intervalo de 0 a 255, pois qualquer valor diferente será reduzido a esse intervalo pela operação $mod256$.

A operação matemática $mod256$ possibilita utilizar o método de análise de freqüência de símbolos para decifrar o código do algoritmo de criptografia posicional.

Na seção 3.2 mostrou-se que deve-se calcular a freqüência relativa dos símbolos tanto no idioma no qual o texto simples foi escrito quanto no texto codificado e após tal análise supor que o símbolo mais freqüente no idioma é o símbolo mais freqüente no texto codificado.

Ao aplicar-se tal método no algoritmo de criptografia posicional deparou-se com o problema de não se conhecer o grau da função de criptografia, como mostrado no seguinte exemplo: para decodificar um símbolo codificado pelo algoritmo posicional usa-se a seguinte expressão $ASCII = (c - f(x)) \pmod{256}$, onde *c* é o valor do símbolo codificado e *ASCII* é o valor do símbolo decodificado; dessa

forma aplica-se o método da análise de freqüência de símbolos para descobrir-se o valor de *c* e o valor de *ASCII* (eles são os símbolos de maior freqüência no texto codificado e no idioma no qual foi escrito o texto simples, respectivamente) e substituindo-se tais valores na fórmula acima obtém-se o valor de $f(x)$. Entretanto deseja-se descobrir a chave de codificação, a qual é composta pelos coeficientes de $f(x)$ e para tal necessita-se do grau de $f(x)$, pois o número de coeficientes de uma função é o número do seu grau mais um.

Como visto no parágrafo anterior, ao aplicar-se o método da análise de freqüência no algoritmo de criptografia posicional, tem-se uma maneira de descobrir-se a imagem de $f(x)$ e a partir deste fato encontra-se a chave de codificação. Para encontrar a chave de codificação monta-se um sistema linear com os valores conhecidos de $f(x)$, obtendo como resultado desse sistema o valor da chave de codificação usada no processo. Ao montar tal sistema necessita-se do grau da função de criptografia, pois ele determina o número de equações que formaram o sistema. Caso contrário tem-se que supor um grau qualquer e resolver o sistema com este suposto grau.

Para montar-se tal sistema linear encontra-se a imagem de $f(x)$ com método da análise de freqüência de símbolos e como sabe-se que o valor de *x* é a posição do caracter no texto, a qual é conhecida, tem-se que os únicos valores desconhecidos são os coeficientes de $f(x)$, os quais objetiva-se descobrir. O exemplo a seguir mostra como pode-se montar tal sistema, supondo que a função de criptografia tem grau dois e portanto a chave de criptografia compõe-se de três valores. Dessa forma deve-se ter um sistema com três equações, assim formadas: a primeira equação é formada tomando como valores de *ASCII* e *c* os símbolos mais freqüentes no idioma no qual foi escrito o texto simples e no texto codificado, respectivamente e obtém-se a primeira imagem de $f(x)$, e a segunda e a terceira equações são formadas de maneira semelhantes, considerando como valores de *ASCII* e *c* o segundo e o terceiro símbolos mais freqüentes tanto no idioma na qual foi escrito o texto simples quanto no texto codificado, respectivamente. O valor de *x* é a posição do símbolo no texto, a qual é determinada pela primeira posição considerada ao iniciar a análise de freqüência no texto codificado, portanto ao começar a análise de freqüência na primeira posição do texto codificado, tem-se que o valor de *x* é igual a um. Dessa forma tem-se o seguinte sistema:

$$\begin{cases} \alpha + \quad \beta + \quad \gamma = c1 - ASCII1 \\ \alpha + \quad \beta + \quad \gamma = c2 - ASCII2 \\ \alpha + \quad \beta + \quad \gamma = c3 - ASCII3 \end{cases}$$

onde α, β e γ são a chave de codificação; c1, c2, c3 são os códigos dos três símbolos codificados mais freqüentes e ASCII1, ASCII2 e ASCII3 são os três símbolos mais freqüentes no idioma no qual escreveu-se o texto simples.

Dessa forma pode-se concluir que a segurança do algoritmo de criptografia posicional é dependente do grau da função de criptografia e uma maneira eficiente de decifrar tal código é saber qual o grau utilizado na função de criptografia.

Capítulo 5

Conclusão

Este trabalho estudou o algoritmo de criptografia posicional com o objetivo de analisar a sua eficiência e segurança. A partir dos estudos realizados pode-se concluir que tal algoritmo apresenta um comportamento linear em função do tamanho do arquivo e tem sua eficiência afetada pelo grau da função de criptografia . Mostrou-se que quanto maior o grau da função de criptografia maior o tempo gasto na codificação do arquivo.

O algoritmo de criptografia RSA apresentou uma complexidade computacional linear nos testes realizados e mostrou-se mais lento que o algoritmo de criptografia posicional em todos os testes executados. Entretanto não se pode afirmar que algoritmo de criptografia posicional sempre apresentará um desempenho melhor que o algoritmo RSA.

O estudo mostrou também que os fundamentos matemáticos necessários ao entendimento do algoritmo posicional são mais simples que os necessários ao algoritmo RSA. Portanto recomenda-se o uso do algoritmo posicional em detrimento ao algoritmo RSA, visto que o mesmo é mais simples e mais rápido que o RSA.

A segurança do algoritmo posicional foi analisada verificando-se a influência da operação matemática $mod256$ na função de criptografia e a possibilidade de decifra-lo através dos métodos da força bruta e de análise de freqüência de símbolos. Tal estudo permitiu verificar que o uso da operação matemática $mod256$ na função de criptografia diminuiu o espaço de busca a ser pesquisado pelo método da força bruta.

Mostrou-se a possibilidade de atacar o algoritmo de criptografia posicional usando-se o método de análise de freqüência de símbolos. Tal método a princípio parecia não ser possível, pois a função de criptografia codifica caracteres iguais em

caracteres distintos. Entretanto a operação matemática $mod256$ associada á ordem seqüencial na qual os caracteres são criptografados permitiu o uso de tal método.

Verificou-se também que a segurança de tal método está fortemente relacionada ao grau da função de criptografia e quanto maior seu grau, maior a sua segurança. Portanto, a maneira mais rápida e eficiente para decifrar a criptografia posicional seria conhecer a priori o grau da função de criptografia, para depois aplicar o método da força bruta ou de análise de freqüência de símbolos. Conhecendo-se o grau da função será suficiente resolver um sistema linear para encontrar a chave de criptografia.

Capítulo 6

Trabalhos Futuros

Dentre os possíveis trabalhos futuros sugere-se:

- analisar a segurança do algoritmo posicional considerando-se o ataque do texto conhecido,
- medir a complexidade do algoritmo posicional em função do grau da função de criptografia. A medida da complexidade em função do grau da chave de criptografia permitiria determinar se em algum momento o tempo de codificação do algoritmo posicional ultrapassa o tempo gasto pelo RSA,
- analisar a complexidade do algoritmo de criptografia RSA para verificar se sua classe de complexidade é a mesma do algoritmo de criptografia posicional,
- introduzir um método que torne aleatória a leitura das posições do texto simples ou a gravação aleatória dos caracteres codificados, afim de evitar o ataque pelo método da freqüência de símbolos e
- analisar a eficiência do algoritmo posicional em relação ao RSA, empregando-se as maiores chaves utilizadas no RSA atualmente. O grau da função de criptografia do algoritmo posicional deve propiciar um espaço de busca equivalente ao RSA considerando-se o método de decifram pela força bruta.
- codificar o texto simples em blocos ao invés de um caracter por vez.
- calcular o módulo da posição do caracter no texto simples antes de substituí-la na função de criptografia.

- analisar o impacto das operações de entrada/saída no tempo de codificação do texto simples.

Referências Bibliográficas

[1] Coutinho. Severino Collier. *Números Primos e Criptografia RSA*. IMPA, 2000.

[2] Moreno. Edward David e Chiaramonte. Rodolfo Barros. Criptografia posicional: Uma solução para segurança de dados, 2001. Artigo publicado na Revista de Iniciação Científica da SBC.

[3] Prazeres. Cássio Vinícius Serafim e Mata Júnior. Gilberto Bittencourt e Reis Júnior. Paulino Batista. Fundamentos teóricos da criptografia, 2000. Monografia de Graduação.

[4] Hemmessy. John L. e Patterson. David A. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1996.

[5] Garfinkel. Simson e Spafford. Gene. *Comércio e Segurança na Web*. Market Books, 1999.

[6] Weber. Raul Fernando. Criptografia contemporânea, 1995. Artigo publicado no sítio www.módulo.com.br.

[7] Ziviani. Nívio. *Projeto de Algoritmos - Com Implementações em Pascal e C*. Editora Pioneira, 1999.

[8] Goldreich. Oded. *Foundations of Cryptograpy - Basic Tools*. Cambridge Univ Press, 2001.

[9] Tanenbaum. Andrew S. *Rede de Computadores*. Editora Campus, 1997.