**UNIVERSIDADE FEDERAL DE SANTA CATARINA**
**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**Mehran Misaghi**

# Avaliação de Modificações do Cifrador Caótico de Roskin

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Prof. Ricardo Felipe Custódio, Dr.
Orientador
custodio@inf.ufsc.br

Florianópolis, Dezembro de 2001

# Avaliação de Modificações do Cifrador Caótico de Roskin

Mehran Misaghi

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Ricardo Felipe Custódio, Dr.
Orientador
custodio@inf.ufsc.br

Prof. Fernando Álvaro Ostuni Gauthier, Dr.
Coordenador do Curso
gauthier@inf.ufsc.br

Banca Examinadora

Prof. João da Silva Dias, Dr.
jdias@ppgia.pucpr.br

Prof. Marcelo da Silva Hounsel, Dr.
dcc2msh@udesc.br

Prof. Vitório Bruno Mazzola, Dr.
mazzola@inf.ufsc.br

*"Não te entristeças salvo por te encontrares longe de Nós;*
*nem exultes, a menos que te estejas aproximando, de*
*regresso a Nós. - Bahá'u'lláh"*

Para minha esposa Patrícia, pela sua compreensão e apoio

e para minha mãe que me observa do Reino de Abhá.

# Agradecimentos

A Deus, que me concedeu a existência. À minha esposa Patrícia pelo seu apoio e incentivo constante e pelos todos os momentos que não fiquei com ela em prol de desenvolvimento este trabalho. À SOCIESC, por ter me dado a oportunidade e apoio para realizar da melhor maneira possível, este trabalho. Ao meu orientador, pelo seu empenho a fim de enriquecer o trabalho. Ao Eduardo da Silva pela ajuda com a edição em LaTeX e na programação em C. Ao Gilsiley Darú pela sua dedicação no desenvolvimento de diversos programas de teste.

# Abstract

A methodology for assessment of Roskin's chaotic cryptosystem is proposed. The principles of cryptography and the principles of chaos related to cryptography were studied and some chaotic cryptosystem were characterizied. For the assessment of chaotic cryptosystems, by means of data compression technique and Strict Avalanche Criterion, some experiments were carried out and the obtained results were analised. According to the experimental results obtained with the cryptosystems cited and discussed in this work, some improvements were proposed. The document intends to become a reference for the evaluation of other chaotic cryptosystems.

# Resumo

Este documento propõe uma metodologia para avaliação do criptossistema caótico de Roskin. Os princípios de criptografia e os princípios de caos relacionados com a criptografia foram estudados e também alguns criptossistemas caóticos foram caracterizados. Para a avaliação de criptossistemas caóticos, através da técnica de compressão de dados e o Critério da Avalanche Estrita, foram realizados diversos experimentos e os resultados obtidos foram analisados. Conforme os resultados dos experimentos obtidos com os criptossistemas citados neste documento, foram propostas melhorias. A intenção é que este documento se torne uma referência para avaliação de outros criptossistemas caóticos.

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

Este documento trata da dissertação de mestrado em Ciência da Computação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

O aumento crescente dos sistemas de informação conectados à rede mundial de computadores tem contribuído na inovação das ferramentas de segurança de dados. Pesquisas recentes junto a 2.700 empresas em todo mundo mostram que 66% dos profissionais de tecnologia de informação consideram a segurança como um dos principais aspectos que devem ser adequadamente tratados. Mas a maioria destas empresas ainda não tem uma política de segurança bem definida [LOP 00].

Uma das mais antigas ferramentas usadas para segurança de dados é a criptografia. O imperador romano Júlio César usava-a quando codificava as mensagens que enviava para Cícero, em Roma, há 2000 anos [MAS 88].

As ferramentas de criptografia providenciam recursos importantes contra acessos intencionais ou acidentais aos dados, os quais podem comprometer sua autenticidade e integridade [STA 99, LEE 99].

A criptografia tem por finalidade receber dados na forma de um texto aberto e produzir uma versão cifrada do mesmo, chamado texto cifrado [CAL 00]. O texto cifrado é um texto não legível para pessoas ou sistemas.

Uma das mais recentes e novas áreas de pesquisa e desenvolvimento

em criptografia é a utilização de sistemas caóticos para cifrar dados. Os sistemas caóticos são caracterizados pela sua sensibilidade as condições iniciais do sistema. Uma pequena variação nos dados de entrada resulta mudanças significativas nos dados de saída.

## 1.1 Objetivos

### 1.1.1 Objetivo Geral

Este trabalho de dissertação tem como objetivo geral propor métodos de avaliação para criptossistemas caóticos.

### 1.1.2 Objetivos Específicos

1. Testar um criptossistema caótico.
2. Propor métodos para avaliação de criptossistemas caóticos.
3. Avaliar a não linearidade de criptossistemas caóticos através:
   (a) Compactação de Dados.
   (b) Critério de Avalanche Estrita.
4. Propor novos criptossistemas caóticos.
5. Propor novos geradores de números randômicos caóticos.

## 1.2 Motivação

Uma das características dos sistemas caóticos é a dependência sensitiva às condições iniciais. Isso pode ser explorado em algoritmos de criptografia, pois uma pequena variação nas condições iniciais (como um bit, por exemplo) resulta em valores bastante diferentes na saída.

Acredita-se que a Teoria do Caos, através desta característica, pode contribuir significativamente no campo da criptografia. Pois a grande alteração no texto cifrado, a partir de uma pequena alteração no texto aberto, é uma característica desejável dos criptossistemas e contribui na redução da vulnerabilidade do criptossistema em questão.

O primeiro trabalho que se tem notícia na área de comunicação de dados baseado na Teoria do Caos foi realizado por Pecora e Carrol, em 1990 [BAP 98].

## 1.3 Materiais e Métodos

Para atender aos objetivos propostos, inicialmente foram feitas as seguintes etapas:

1. Estudar fundamentos de criptografia.
2. Estudar princípios da teoria de caos.
3. Estudar criptossistemas caóticos.
4. Estudar geradores de números randômicos.
5. Estudar geradores de números randômicos caóticos.

Para atender o objetivo 3, foram desenvolvidos alguns programas em linguagem de programação C, cujos códigos fontes encontram-se no apêndice B.

Para atender ao objetivo 3a, escrito na página 15, foram criados diversos modelos de textos abertos, com o intuito de englobar a maior variedade de textos abertos existentes.

Os experimentos realizados referente ao objetivo 3b, foram baseados em programa **sac.c** e, o código fonte também encontra-se no anexo B. Os dados estatísticos resultantes permitiram a criação de diversas figuras que encontram-se no capítulo 5.

Os programas utilizados para realização de experimentos foram desenvolvidos em linguagem de programação C. Foram utilizados equipamentos com sistema

operacional Windows NT para execução destes programas e sistema operacional Conectiva Linux versões 4.2 e 5[1] para escrever o documento.

Esta dissertação foi escrita utilizando LaTeX no editor de textos *vim*[2]. O programa `ispell`[3] foi utilizado para a correção ortográfica. As figuras foram feitas no `Corel Draw`[4] e exportadas para o formato `eps`.

A conversão do documento LaTeX para os formatos `ps` e `pdf` foi feita através das ferramentas `dvips` e `ps2pdf`. Para visualização e impressão dos documentos em `ps` e `pdf`, foi utilizado o programa `gv`[5].

## 1.4 Conteúdo do Documento

O presente documento está estruturado da seguinte forma: O capítulo 2 descreve os princípios da criptografia que serve como base para outros capítulos, para identificar as características dos criptossistemas caóticos. A Teoria do Caos, bem como sua terminologia é estudada no capítulo 3, que verifica a aplicação de Caos em Criptografia e também são citados alguns cifradores que utilizam equações caóticas, entre eles, os cifradores de Roskin [ROS 99].

O capítulo 4 além de descrever números pseudo-randômicos e os geradores de números pseudo-randômicos, apresenta os geradores de números pseudo-randômicos caóticos, exemplificando alguns casos. O capítulo 5 avalia os criptossistemas de Roskin e descreve os experimentos realizados com tais criptossistemas, referente à compactação e ao efeito de avalanche estrita. Este capítulo também descreve os métodos utilizados e conclui com os resultados obtidos para diversos modelos propostos.

Baseando-se nos resultados obtidos no capítulo 5, foram propostos dois modelos de criptossistemas caóticos no capítulo 6. Para verificar se os modelos propostos contribuem na redução de vulnerabilidade do criptossistema de Roskin, os experimentos

[1] `http://www.conectiva.com.br`
[2] `http://www.vim.org`
[3] `http://fmg-www.cs.ucla.edu/ficus-members/geoff/ispell.html`
[4] `http://www.corel.com`
[5] `http://wwwthep.physik.uni-mainz.de/~plass/gv/`

referentes à compactação e ao efeito avalanche estrita foram repetidos com estes modelos. Este capítulo termina apresentando os resultados obtidos com tais experimentos. E por fim, o capítulo 7 apresenta os objetivos propostos que foram alcançados e analisa de forma resumida os resultados obtidos.

Os apêndices contêm os códigos fontes de todos os programas que foram desenvolvidos e utilizados neste documento. O apêndice A apresenta os códigos fontes dos criptossistemas de Roskin. O apêndice B apresenta os códigos fontes dos programas utilizados para realização dos experimentos e criação dos modelos para compactação e o efeito de avalanche estrita. O código fonte do cifrador do Rijndael, que foi utilizado para comparação de resultados de compactação, encontra-se no apêndice C. O apêndice D apresenta códigos fontes dos dois modelos, que foram propostos no capítulo 6. Os resultados dos experimentos realizados com os modelos propostos estão no apêndice E.

# Capítulo 2

# Princípios da Criptografia

## 2.1 Introdução

Este capítulo define a terminologia utilizada em criptografia, conceitua os seus serviços, descreve os tipos de cifradores e faz uma comparação entre criptografia simétrica e assimétrica.

Para poder avaliar as características de criptossistemas, também são estudados neste capítulo os métodos de difusão e confusão, transposição e substituição, características e tipos de chaves e Efeito de Avalanche Estrita.

No final deste capítulo, está descrita a tendência dos métodos atuais da criptografia.

## 2.2 Terminologia Básica

**Texto Aberto** é qualquer informação legível. **Cifrar** é o processo de codificar ou esconder o conteúdo de um texto aberto. **Cifrador** é o algoritmo que transforma o texto aberto em um texto cifrado através de diversos métodos.

A palavra **criptografia** é a junção das palavras gregas $\kappa\rho\upsilon\pi\tau\sigma\varsigma$ (kryptos - secreto) e $\gamma\rho\alpha\phi\omega$ (grapho - escrita) [dC 00], ou seja, escrita secreta.

**Criptanálise** é a ciência que estuda as técnicas para obter o texto aberto

a partir do texto cifrado [SCH 98].

Um texto aberto pode ser um bloco de bits, um arquivo de imagem, entre outros, e é representado por $P$ (Plain). O texto cifrado é representado por $C$ (Cipher). O processo que produz o texto cifrado é representado por $E$ (Encryption) e o processo que decifra o texto cifrado é representado por $D$ (Decryption). A chave utilizada é representada por $K$ (Key). Por exemplo, a figura 2.1 mostra que Alice para enviar um texto aberto $P$ , para Bob, utiliza chave $K$ para cifrar. Bob através da mesma chave $K$ , que é disponível para ambos, decifra o $C$ e recupera o $P$ .

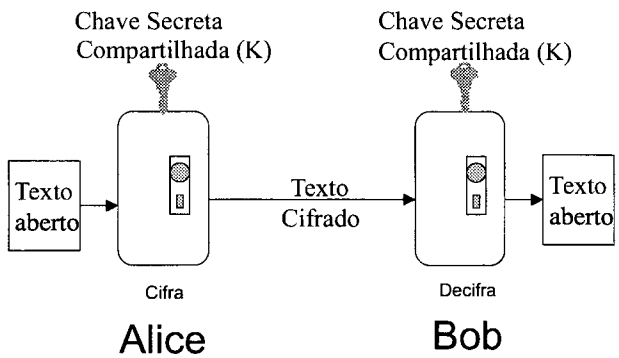


Figura 2.1: Modelo simplificado de criptografia convencional. Alice utiliza chave compartilhada $K$ para cifrar o texto aberto. O Bob através da mesma chave compartilhada $K$, decifra o texto cifrado e recupera o texto aberto.

## 2.3 Serviços de Criptografia

A Criptografia fornece às pessoas devidamente autorizadas, serviços que asseguram confidencialidade, autenticação, integridade e não repudiação para dados [STA 99, LEE 99], conforme definido abaixo:

***Confidencialidade***

Consiste em restringir o acesso aos dados somente às pessoas autorizadas.

***Autenticação***

A autenticação estabelece a autenticidade e validade da transmissão, da mensagem e das entidades envolvidas. Assegura também que as entidades envolvidas (emissor e receptor da mensagem) sejam autênticas.

***Integridade***

O objetivo deste serviço é garantir que os dados recebidos sejam idênticos aos dados enviados e que não sofram nenhuma alteração acidental ou não autorizada, incluindo inserção, exclusão e modificação de conteúdo.

***Não Repudiação***

É o serviço que evita as ações individuais do emissor e receptor de mensagem quanto à negação de envio ou recebimento. Evita que uma mensagem enviada, produzida ou recebida pelo $A$ seja negada por mesmo.

## 2.4 Criptografia Simétrica e Criptografia Assimétrica

### 2.4.1 Criptografia Simétrica

**A Criptografia Simétrica**, também chamada de **Criptografia Convencional** ou **Criptografia de Chave Secreta** [SCH 98, STA 99], utiliza a mesma chave para cifrar e decifrar as mensagens. Isto implica em um acordo entre emissor e receptor na utilização da mesma chave e mesmo algoritmo, antes do início do envio e recebimento das mensagens, conforme figura 2.1.

O emissor para cifrar o texto aberto $P$ utiliza a função de criptografia $E$ com a chave $K$, gerando o texto cifrado $C$ e envia-o para o receptor desta forma:

$$C = E_K(P) \tag{2.1}$$

Se o receptor tiver a chave $K$, poderá decifrar o texto cifrado $C$ da seguinte forma:

$$P = D_K(C) \tag{2.2}$$

A segurança da criptografia simétrica depende do tamanho da chave utilizada, de forma que quanto maior for o tamanho da chave, o criptossistema será menos vulnerável. Uma vez descoberta a chave, qualquer um pode cifrar e decifrar as mensagens.

Quando somente dois usuários estão se comunicando, precisa-se apenas de **1** chave. Se três usuários desejarem comunicar entre si, há necessidade de **3** chaves. Esta necessidade aumenta na medida que cresce o número de usuários que estão se comunicando. A quantidade de chaves necessárias pode ser apresentada através da seguinte equação:

$$Quantidade\ de\ chaves = \frac{n(n-1)}{2} \tag{2.3}$$

A figura 2.2 ilustra a necessidade de **15** chaves, para que os **6** usuários da rede se comuniquem [MEN 96].

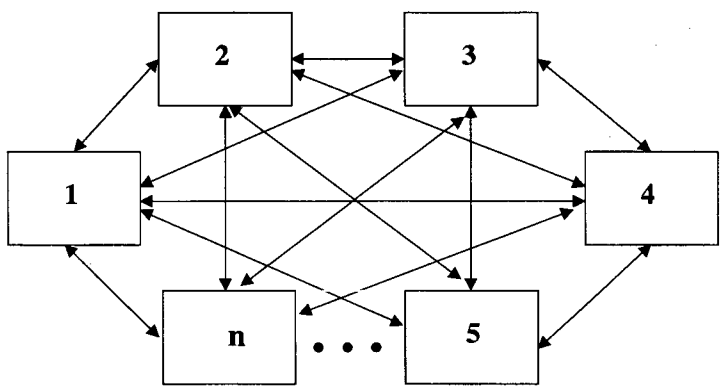


Figura 2.2: Volume de chaves para criptossistemas simétricos com 6 usuários.

Um criptossistema simétrico bem conhecido é o Data Encryption Standard (DES) que foi introduzido em 1977, com chave de 56 bits e bloco de 64 bits [LEN 99]. Como outros exemplos de criptossistemas simétricos pode-se citar:

1. 3-DES com chave de 112 bits e bloco de 64 bits.

2. IDEA com chave de 128 bits e bloco de 64 bits.

3. RC5 com chave e bloco de tamanho variável.

### 2.4.2 Criptografia Assimétrica

Na **Criptografia Assimétrica**, também chamada de **Criptografia de Chave Pública**, utiliza-se duas chaves matematicamente relacionadas. Uma chave **pública** e uma outra diferente chave **privada**. Se é utilizada a **chave pública** para cifrar, deve-se utilizar a **chave privada** para decifrar, e vice-versa.

Imagina-se que o emissor A queira enviar algo para o receptor B[1]. B tem duas chaves. Uma chave pública $KU_b$, que é conhecida por A, e uma chave privada $KR_b$, que somente ele tem conhecimento.

Para cifrar um texto aberto P = $\{P_1, P_2, ..., P_M\}$ para ser enviado para B, pode-se utilizar a chave pública de B:

$$C = E_{KUb}(P) \tag{2.4}$$

B, ao receber o texto cifrado C, consegue obter o texto aberto P através da sua chave privada, $KR_b$:

$$P = D_{KRb}(C) \tag{2.5}$$

O processo de criptografia assimétrica pode ser visualizado na figura 2.3 que ilustra os passos de criptografia assimétrica, segundo Stallings [STA 99]:

1. Cada usuário gera um par de chaves que será utilizado para cifrar e decifrar.

2. Cada usuário disponibiliza a sua chave pública em um local público, para que todos tenham acesso.

3. A chave privada do usuário deve ser mantida em um local seguro, de forma que somente o próprio usuário tenha acesso.

4. Quando A deseja enviar uma mensagem para B, ela utiliza a chave pública do B para cifrar a mensagem.

[1] A é abreviatura de Alice e B é abreviatura de Bob.

5. Quando B recebe esta mensagem, ele poderá decifrá-la utilizando a sua chave privada. Ninguém poderá decifrar esta mensagem sem ter conhecimento da chave privada de B.

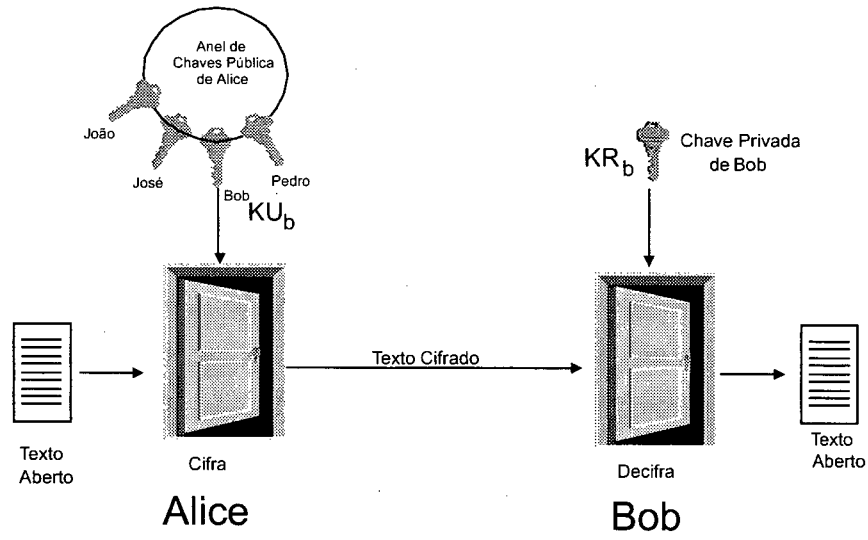


Figura 2.3: Criptografia Assimétrica. Quando **Alice** deseja enviar uma mensagem para **Bob**, ela utiliza a chave pública do **Bob**, $KU_b$, para cifrar a mensagem. Quando **Bob** recebe esta mensagem, ele poderá decifrá-la utilizando a sua chave privada, $KR_b$.

Segundo [STA 99], primeiro criptossistema assimétrico funcional foi proposto por Clifford Cocks em 1973. Em 1977 Ron Rivest, Adi Shamir e Len Adleman desenvolveram o algoritmo de RSA[2].

### 2.4.3 Criptografia Simétrica X Criptografia Assimétrica

Uma vantagem da **criptografia assimétrica** é a não necessidade de duas entidades compartilharem uma chave privada, como ocorre com a **criptografia simétrica**.

[MEN 96, SCH 98, STA 99] comparam criptografia simétrica e criptografia assimétrica, citando suas vantagens e desvantagens:

[2]Rivest-Shamir-Adleman

- **Vantagens da Criptografia Simétrica**

1. A criptografia simétrica pode ser utilizada para obter altas taxas de transferência de dados.

2. As chaves utilizadas para criptografia simétrica são relativamente pequenas.

3. Os cifradores simétricos podem ser empregados como primitivos para construir vários mecanismos criptográficos, como geradores de números pseudo-randômicos e assinaturas digitais eficientes.

- **Desvantagens da Criptografia Simétrica**

1. Em uma rede relativamente grande, existem vários pares de chaves para serem gerenciados. Conseqüentemente, precisa-se de um esquema eficiente de gerenciamento de chaves.

2. Em uma comunicação entre duas entidades, a chave deve permanecer secreta em ambas as partes.

3. Não é possível autenticar uma assinatura digital recebida via criptografia simétrica. Para autenticá-la, é necessário empregar chaves relativamente grandes.

4. Em uma comunicação entre duas entidades, as chaves utilizadas devem ser trocadas a cada nova seção.

- **Vantagens da Criptografia Assimétrica**

1. Somente a chave privada deve ser mantida segura. A autenticidade das chaves públicas, entretanto, deve ser garantida.

2. A autoridade de chave pública é responsável pelo gerenciamento, armazenamento e distribuição das chaves. Assim não há necessidade de cada entidade armazenar diversas chaves para estabelecer a comunicação com outras entidades.

3. Dependendo do modo de utilização, o par de chaves (privada e pública) pode permanecer inalterado por um longo período de tempo.

4. Muitos algoritmos de criptografia assimétrica fornecem mecanismos eficientes de assinatura digital. A chave utilizada neste caso específico é menor em relação à chave utilizada na criptografia simétrica.

5. Em uma rede grande, o número de chaves necessárias pode ser consideravelmente menor em relação à criptografia simétrica no mesmo cenário.

- **Desvantagens da Criptografia Assimétrica**

1. As taxas de transferência de dados para métodos de criptografia assimétrica são muito menores, comparadas às da criptografia simétrica.

2. O tamanho das chaves na criptografia assimétrica é muito maior do que na criptografia simétrica.

- **Qual modelo de criptografia deve ser utilizado?**

Segundo Lee, [LEE 99], a **criptografia assimétrica** é adequada para entidades multi-usuários que têm comunicação com outras entidades externas, onde há necessidade da autenticação dos usuários e das mensagens. A **criptografia simétrica** pode ser empregada onde não há comunicação entre outras entidades externas e uma única autoridade conhece e gerencia todas as chaves utilizadas.

Os mecanismos da **criptografia assimétrica** podem ser utilizados para cifrar uma chave privada, sendo que a mesma pode ser utilizada para cifrar uma mensagem ou um arquivo.

## 2.5 Tipos de Cifradores

### 2.5.1 Cifradores de Fluxo

Os **cifradores de fiuxo** cifram o texto aberto bit a bit. Estes cifradores utilizam alguma variação do algoritmo **One Time Pad** [SCH 98], cujo processo está descrito a seguir:

Supondo que o texto aberto $P$ tem $n$ bits (n¿ 0), então:

$$P = (p_1, p_2, p_3, ..., p_n) \tag{2.6}$$

e uma chave de $n$ bits:

$$K = (k_1, k_2, k_3, ..., k_n) \tag{2.7}$$

o texto cifrado será:

$$C = (c_1, c_2, c_3, ..., c_n) \tag{2.8}$$

Para cifrar, utiliza-se a operação **OU EXCLUSIVO** entre o texto aberto e a chave em questão.

$$c_i = p_i \bigoplus k_i \tag{2.9}$$

Para decifrar, basta realizar novamente a operação **OU EXCLUSIVO**, com o texto cifrado e a mesma chave que foi utilizada para cifrar o texto aberto.

$$p_i = c_i \bigoplus k_i \tag{2.10}$$

Este algoritmo é teoricamente inquebrável, segundo Schneier [SCH 98], uma vez que seja utilizada uma chave randômica e que esta não seja reutilizada.

O esquema original do **One Time Pad** não é prático devido ao tamanho da chave, que deve ser igual ao tamanho do texto e também não poderá ser reutilizada.

Na prática, utiliza-se um gerador de números randômicos que serve como semente para entrada do algoritmo **One Time Pad**.

### 2.5.2 Cifradores de Bloco

Cifradores de Bloco geralmente operam em blocos de texto, onde o comprimento do texto cifrado é igual ao do texto aberto, utilizando a mesma chave [SCH 98].

Para cifrar um texto aberto com cifradores de bloco, quebra-se este texto em diversos blocos e utiliza-se uma chave $K$ para cifrar os blocos. Para decifrar o texto cifrado, basta passar para o cifrador com a mesma chave utilizada.

### 2.5.3 Cifradores de Fluxo X Cifradores de Bloco

A diferença básica entre cifradores de fluxo e de bloco é na sua forma de implementação. A implementação de cifradores de fluxo é mais apropriada para componentes eletrônicos que transformam os dados bit a bit. Os cifradores de bloco são mais adequados para implementações de software, onde um conjunto de bits faz mais sentido do que um bit [SCH 98].

## 2.6 Tipos de Cifradores de Bloco

Duas classes importantes de cifradores de bloco são **Cifradores de Substituição** e **Cifradores de Transposição** [MEN 96]. Alguns algoritmos dispõe dos dois cifradores ou de uma combinação entre os dois [MEN 96, SCH 98].

### 2.6.1 Cifradores de Substituição

Em um **Cifrador de Substituição**, cada caracter no texto aberto é substituído por outro caracter no texto cifrado. Para recuperar o texto aberto, o receptor inverte a substituição no texto cifrado [SCH 98]. Segundo [SCH 98], pode-se encontrar quatro tipos de cifradores de substituição:

- **Cifradores de Substituição Simples** ou **Cifrador Monoalfabético**

Cada caracter do texto aberto é substituído por um caracter correspondente do texto cifrado. Como exemplo, pode ser citado o cifrador de César que envolve a substituição de cada letra do alfabeto com a letra que fica três posições a partir desta letra [STA 99]. A tabela 2.1 mostra o cifrador de César.

Tabela 2.1: Cifrador de César

| P | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c |

Atribuindo um valor numérico para cada letra (a=1, b=2,..., z=26), pode-se ter a seguinte notação para o cifrador de César:

$$c = E(p) = (p + 3)mod(26) \tag{2.11}$$

$$p = D(c) = (c - 3)mod(26) \tag{2.12}$$

- **Cifradores de Substituição Homofônico**

É parecido com o cifrador de substituição simples, exceto que cada caracter do texto aberto pode ser mapeado para um dos diversos caracteres do texto cifrado.

Segundo Schneier, [SCH 98], os cifradores de substituição homofônicos foram utilizados em torno de 1401 pelo Duque de Mantua. Estes cifradores eram mais complicados para quebrar do que os cifradores de substituição simples.

- **Cifradores de Substituição Múltipla**

Segundo Schneier, [SCH 98], estes cifradores cifram um conjunto de caracteres em um grupo cifrado. Por exemplo TIME, pode corresponder a RTRD.

Segundo [STA 99], o mais conhecido cifrador de substituição múltipla é o cifrador **Playfair** que foi inventado por Sir Charles Wheatstone, cientista inglês, em 1854. Este cifrador trata os digramas no texto aberto como unidades simples e traduz estas unidades nos digramas do texto cifrado.

- **Cifradores de Substituição Polialfabético**

É um conjunto de múltiplos cifradores de substituição simples que foi inventado por Leon Battista, em 1568 [SCH 98]. Um cifrador de substituição polialfabético tem os seguintes recursos [STA 99]:

1. Um conjunto de regras referente à substituição monoalfabética são utilizados.

2. Uma chave determina qual regra particular é escolhida para uma certa transformação.

Estes cifradores foram utilizados durante a Guerra Civil Americana pelo Exército da União [SCH 98].

Como outro exemplo de cifradores de substituição polialfabético, pode-se citar o cifrador Vigenère [STA 99].

### 2.6.2 Cifradores de Transposição

Em um **Cifrador de Transposição**, o texto aberto permanece o mesmo, mas a ordem de caracteres é trocada totalmente [SCH 98].

Em um **Cifrador de Transposição Colunar**, o texto aberto é escrito horizontalmente, em uma folha com margens fixas e o texto cifrado é lido verticalmente [SCH 98]. Por exemplo, para cifrar o texto: **SEGURANÇAIMPLICANAESCOLHADEBOACHAVE** em uma tabela com **7** colunas, começa-se a preencher a tabela na primeira linha e primeira coluna horizontalmente até que todos os caracteres sejam colocados na tabela. Assim, completa-se a tabela com **5** linhas. Para obter o texto cifrado, faz-se a leitura vertical desta tabela, obtendo-se o texto: **SÇCOOEAALAGINHCUMAAHRPEDAALSEVNICBE**. A tabela 2.2 ilustra este exemplo.

Tabela 2.2: Cifrador de Transposição Colunar

| | | | | | | |
|---|---|---|---|---|---|---|
| *S* | *E* | *G* | *U* | *R* | *A* | *N* |
| *Ç* | *A* | *I* | *M* | *P* | *L* | *I* |
| *C* | *A* | *N* | *A* | *E* | *S* | *C* |
| *O* | *L* | *H* | *A* | *D* | *E* | *B* |
| *O* | *A* | *C* | *H* | *A* | *V* | *E* |

O cifrador alemão ADFGVX, utilizado durante a Primeira Guerra Mundial, é um cifrador de transposição combinado com um cifrador de substituição simples

[SCH 98].

## 2.7 Segurança de um Criptossistema

Segundo Schneier, [SCH 98], a segurança de um criptossistema depende das respostas às seguintes perguntas:

1. Quanto são valiosos os dados de entrada neste criptossistema?
2. Por quanto tempo precisa-se de segurança?
3. Quais são os recursos dos adversários?

Para aumentar a segurança de um criptossistema são utilizados diversos métodos. As seções 2.7.1 e 2.7.2 descrevem estes métodos.

### 2.7.1 Difusão e Confusão

Os métodos de *Difusão* e *Confusão* foram introduzidos por Claude Shannon para dificultar a descoberta do texto aberto a partir do texto cifrado [STA 99].

*Difusão*: Segundo Massey, [MAS 88], a estrutura estatística do texto aberto deve ser dissipada no texto cifrado.

*Confusão*: Este método objetiva tornar a relação entre o texto cifrado e a chave tão complexa quanto possível, para evitar a descoberta da chave.

### 2.7.2 Critério de Avalanche Estrita

Uma propriedade desejável para qualquer criptossistema é que uma pequena mudança no texto aberto ou na chave deveria produzir uma mudança significativa no texto cifrado [STA 99].

Quando a mudança de um bit no texto aberto ou na chave resulta em mudança de muitos bits no texto cifrado, pode-se dizer que o criptossistema em questão possui um **Efeito Avalanche** [STA 99].

Segundo [SEB 94, KIM 91], Websters e Tavares introduziram o **Critério de Avalanche Estrita**. O Critério de Avalanche Estrita (Strict Avalanche Criterion) ou simplesmente SAC ocorre quando a mudança de um único bit no texto aberto ou na chave resulta na mudança de, exatamente, metade dos bits do texto cifrado.

### 2.7.3 Tamanho da Chave para Cifradores de Bloco

A tabela 2.3 mostra os tamanhos das chaves recomendadas em uma aplicação de criptografia para cifradores de bloco [RDS 98]. O tamanho da chave utilizada nas aplicações de criptografia é relativo ao nível da aplicação. Quanto mais valiosa for a informação, é recomendado utilizar chaves de tamanhos maiores.

Tabela 2.3: Chaves recomendadas para cifradores de bloco

| Nível | Tamanho | Recomendado para |
|---|---|---|
| **Exportação** | 40 bits | Exportação (pouca proteção) |
| **Pessoal** | 56/64 bits | Aplicações pessoais que não precisam de muita proteção |
| **Comercial** | 128 bits | Informação valiosa, mas não extremamente sensível |
| **Militar** | 160 bits | Autoridades de Certificação<br>Informação valiosa e extremamente sensível |

## 2.8 O Estado da Arte

Os métodos existentes de criptografia, atualmente, estão empregados em diversos segmentos do mercado. Com o aumento constante na tecnologia de computação, aumento acelerado de sistemas interconectados e avanço significativo em poder computacional, novos e poderosos métodos de criptografia são necessários e também viabilizados [LEE 99].

- **DES - Um Primeiro Padrão de Criptografia**

Por mais de vinte anos, o DES foi empregado amplamente como padrão de criptografia em diversos sistemas. Mas com os atuais recursos computacionais, o DES não pode ser considerado suficientemente seguro pois tornou-se obsoleto. Foram feitas diversas tentativas de quebra do DES, através de implementações de *hardware* e *software* [LEN 99].

Em 1977, uma máquina de busca paralela, com custo de 20 milhões de dólares, foi proposta com um tempo estimado de 12 horas de pesquisa. Em 1980, este valor foi corrigido para 50 milhões de dólares e 2 dias de pesquisa. Em 1993, este custo baixou para um milhão de dólares e 3,5 horas de pesquisa. Em 1998, um máquina de 130.000 dólares foi montada com uma expectativa de tempo de pesquisa de 112 horas [LEN 99].

O tamanho da chave do DES para fora dos Estados Unidos até 1999 era de 40 bits. Atualmente, é possível calcular a chave correta em torno de 18 minutos, com 1 bilhão de tentativas de chaves por segundo [TER 00].

- **AES - Novo Padrão de Criptografia**

Em 1997, o National Institute of Standards and Tenchnology (NIST) dos E.U.A. abriu uma competição internacional chamada AES (Advanced Encryption Standard) para algoritmos candidatos que satisfizessem os seguintes critérios [TER 00, LEE 99]:

1. Tamanho de bloco 128 bits na entrada e na saída.

2. Tamanho da chave de 128 ou 192 ou 256 bits.

3. Segurança e velocidade do algoritmo igual ou superior a 3-DES.

4. Implementação em *software*, *hardware* e *smart-card* de forma eficiente.

5. Código fonte do algoritmo disponível sem custo algum.

Em abril de 1999, foram escolhidos cinco candidatos [FOR 00] que são os seguintes:

1. **MARS**, desenvolvido por Nevenko Zunic (IBM).
2. **RC6**, desenvolvido por Burt Kaliski (RSA Laboratories).
3. **RIJNDAEL**, desenvolvido por Joan Daemen e Vincent Rijmen.
4. **SERPENT**, desenvolvido por Ross Anderson, Eli Biham e Lars Knudsen.
5. **TWOFISH**, desenvolvido por Bruce Schneier, J. Kelsey, D. Whiting, D. Wagner, Chris Hall e Niels Ferguson.

Em 2 de outubro de 2000, foi escolhido RIJNDAEL como novo padrão de criptografia que substitui oficialmente o DES, a partir de junho de 2001.

Quanto à questão de segurança do AES, com 128 bits, baseado em [FOR 00], presumindo que alguém monte uma máquina que consiga uma chave correta do DES em apenas um segundo, esta máquina levará 149 trilhões de anos para conseguir uma chave do AES de 128 bits ou, na prática, menos de 20 bilhões de anos.

## 2.9 Conclusão

Neste capítulo foi apresentado a terminologia utilizada em criptografia, bem como as técnicas e métodos empregados na construção de criptossistemas. Foram explicados, também, diversos tipos de criptografia.

# Capítulo 3

# Princípios do Caos

## 3.1 Introdução

Este capítulo aborda os conceitos preliminares da **Teoria do Caos** e discute a aplicação de caos em criptografia. As características dos criptossistemas caóticos são explicadas. Os cifradores caóticos também são exemplificados.

## 3.2 Teoria do Caos

### 3.2.1 Definição

Inicialmente, a palavra caos pode nos lembrar qualquer desordem que produz confusão em nossa mente [MEI 00]. Mas, na verdade, a palavra **caos** se origina da palavra grega **khaos**. Falando de uma forma simples, caos é obter resultados totalmente aleatórios a partir de equações simples e normais, ou seja, existe ordem nos dados que parecem ser completamente aleatórios [RAE 98].

A **Teoria do Caos** é definida como o estudo de sistemas não lineares complexos, ou seja, o estudo de sistemas em constante alteração baseado em conceitos matemáticos e que representam o comportamento de um sistema em determinados momentos [HO 95].

A **Teoria do Caos** mostra que os sistemas não lineares, que não são pre-

visíveis e são caracterizados através de equações diferenciais, podem se tornar previsíveis através da representação de comportamento e estado do sistema, utilizando equações caóticas.

A **Teoria do Caos** pode ser considerada como uma nova forma de interpretar dados científicos para obter resultados mais precisos e mais naturais [HO 95].

Segundo Kung, [YAO 98], a **Teoria do Caos** tem oferecido excelentes explicações para diversos fenômenos naturais complexos.

### 3.2.2 Determinismo e Sistemas Lineares

Existe uma crença filosófica há milhares de anos, referente ao mundo material, de que qualquer evento é resultado inevitável de eventos precedentes. Assim, todos os eventos são totalmente previsíveis em todos os momentos futuros e passados [TRU 98].

Há 300 anos, na Inglaterra, Isaac Newton introduziu os princípios do Determinismo e os Sistemas Lineares na ciência moderna. Ele descobriu um conjunto de leis que podiam prever o movimento e comportamento de diversos sistemas com alto grau de precisão [TRU 98].

As leis de Newton são determinísticas, pois através de equações matemáticas demonstram que qualquer acontecimento no futuro é completamente determinado por fatos que acontecem no presente. Os acontecimentos atuais são completamente determinados por acontecimentos em qualquer tempo no passado [TRU 98].

### 3.2.3 Condições Iniciais

Empregando parâmetros apropriados para um dado sistema, os valores destes para o momento em que o sistema está começando é chamado de **condições iniciais** do sistema em questão [TRU 98].

As leis de Newton são determinísticas, pois elas implicam em que, para qualquer sistema, as mesmas condições iniciais sempre produzirão os mesmos resultados de forma idêntica [TRU 98].

### 3.2.4 Incerteza

Como no mundo real os instrumentos utilizados para medição (mesmo os mais perfeitos e mais avançados) não possuem precisão infinita, deve-se incluir o parâmetro da **incerteza** nos sistemas em estudo.

Em dinâmica, a incerteza existente em qualquer sistema indica que as condições iniciais não podem ser especificadas com uma exatidão infinita.

Uma forma de diminuir a incerteza nos resultados finais em dinâmica é aumentar o grau de exatidão na medição das condições iniciais.

### 3.2.5 Caos

Poincaré [TRU 98], em seus estudos baseados nas leis de Newton, que tinham como objetivo descrever equações matemáticas para o movimento de planetas, sentiu a necessidade de incluir pequenas incertezas para aumentar a exatidão nos resultados finais. Isto porque era praticamente impossível medir com precisão a posição e a velocidade inicial dos planetas.

Poincaré descobriu que certos sistemas não obedecem às leis determinísticas, isto é, a alteração de valores iniciais não gera resultados alterados na mesma proporção. Para estes sistemas, ele demonstrou que uma pequena imprecisão nas condições iniciais geraria dois resultados completamente diferentes [TRU 98].

Para estes sistemas, uma pequena imprecisão ou incerteza nas condições iniciais, faz com que os resultados finais sejam bastante diferentes e não previsíveis como as leis determinísticas. Então, os resultados finais destes sistemas dependem muito das condições iniciais. A **dependência sensitiva** ou **sensibilidade** às condições iniciais [FER 94], presente nos sistemas que foram estudados por Poincaré, é chamada **Instabilidade Dinâmica** ou simplesmente **Caos** [TRU 98].

A presença do comportamento caótico em outros sistemas foi inicialmente observada por um meteorologista chamado Edward Lorenz, em 1960. Lorenz, através de seu computador e um conjunto de doze equações, fazia previsões meteorológicas. Os resultados sempre eram os mesmos quando as condições iniciais se re-

petiam.

Através de uma pequena variação nas condições iniciais em um dos experimentos (0,506127 ao invés de 0,506), Lorenz obteve resultados totalmente diferentes. Assim, ele concluiu que as previsões não eram exatas devido às imprecisões existentes na determinação das condições iniciais [FER 94].

Este efeito é conhecido por **Efeito Borboleta**, pois a diferença entre as condições iniciais de duas curvas é tão pequena que pode ser comparada à uma borboleta que está movimentando as suas asas. A figura 3.1 ilustra o Efeito Borboleta. Segundo Gleick, [GLE 00], esta imagem que é parecida às das asas de uma borboleta, tornou-se um emblema para os primeiros investigadores de caos. Como sistema nunca se repete, a trajetória cruza lugares diferentes e forma movimentos circulares que são simétricos e inversos.

Pequenas alterações nas condições meteorológicas implicam em tempestades e nevascas. Os erros e incertezas se multiplicam e formam "um efeito de cascata ascendente através de uma cadeia de aspectos turbulentos, que vão dos demônios da poeira e tormentas até redemoinhos continentais que só os satélites conseguem ver " [GLE 00].

Em 1963, Lorenz [DRA 94] publicou um artigo descrevendo o que tinha acontecido com as suas equações, incluindo a não previsibilidade meteorológica e discutiu os tipos de equações que causam estes comportamentos.

### 3.2.6 Função Cáotica

A função caótica mais conhecida (chamado também de Mapa Logístico) pode ser definida desta forma [FER 94]:

$$X_{i+1} = rX_i(1 - X_i) \tag{3.1}$$

onde $\{X_i, X_{i+1} \in R \ e\, 0 \leq X_i, X_{i+1} \leq 1\}$ e $\{r \in R, 0 \leq r \leq 4\}$

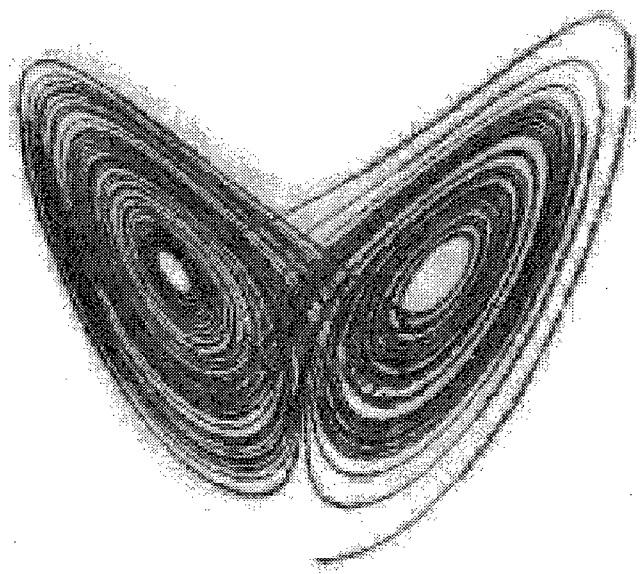

Figura 3.1: Efeito Borboleta.

Nesta equação $i$, é uma variável que muda a cada iteração, e $r$ é um parâmetro fixo que é dado no início de iterações e não muda mais.

## 3.3 Aplicação de Caos nas Diversas Ciências

A presença de sistemas caóticos na natureza contradiz a previsibilidade das leis determinísticas e diminui o grau de certeza dos resultados gerados por estes sistemas.

A **Teoria do Caos**, como uma nova ciência, é objeto de estudo em diversas instituições de ensino e pesquisa. O caos está presente em sistemas biológicos, bioquímicos e de mercado; em fractais e na sincronização de sinais, entre outros [TRU 98].

## 3.4 Características dos Sistemas Caóticos

Ross [ROS 96] comenta as principais características para sistemas caóticos:

1. Os sistemas caóticos são determinísticos. Isto quer dizer que eles possuem algo determinando o seu comportamento.

2. Os sistemas caóticos são muito sensíveis às condições iniciais. Uma pequena alteração no seu estado inicial pode resultar em dados de saída extremamente diferentes. Isto torna o sistema proporcionalmente imprevisível.

3. Os sistemas caóticos aparentam ser desordenados, até randômicos. Mas eles não são. Além de comportamento randômico eles possuem um sentido de ordem e modelo. Na verdade, sistemas randômicos não são caóticos.

## 3.5 Como o caos pode ser utilizado na criptografia?

A **dependência sensitiva** que foi discutida anteriormente pode contribuir significativamente na criptografia.

Quando um texto é cifrado, o criptoanalista deseja descobrir, de alguma forma, uma pequena seqüência de bits através da variação de dados de entrada ou a variação da chave utilizada.

Se o criptossistema utilizado possui alguma função caótica, a variação de apenas um bit na entrada dos dados neste criptossistema resulta em dados cifrados completamente diferentes. Desta forma, se um criptoanalista está tentando decifrar dados, utilizando as mesmas condições iniciais com pequena variação e procurando modelos para obter a chave, ele dificilmente conseguirá obter dados corretos [ROS 99].

Segundo [INS 98], os criptossistemas caóticos propostos na literatura têm as seguintes características:

1. A função caótica é simples, mas os sinais e resultados gerados são bastante complexos.

2. É fácil utilizar a função caótica sem ter risco de cálculos que geram *overflow*.

3. É muito difícil para o criptoanalista obter resultados adequados com o criptossistema que utiliza a propriedade de dependência sensitiva às condições iniciais.

4. Não é previsível.

Levando em consideração as características acima citadas, será extremamente difícil descobrir o texto cifrado e praticamente impossível detectar a chave utilizada [INS 98].

## 3.6 Diagrama de um Criptossistema Caótico

A figura 3.2 apresenta o diagrama típico de um criptossistema caótico, baseado no que consta em [WU 99, YAN 97].

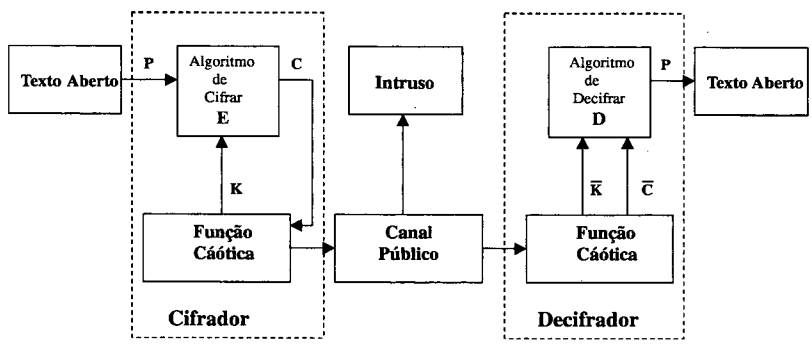


Figura 3.2: Diagrama típico de um Criptossistema Caótico. O **Cifrador** é composto por uma função caótica, um algoritmo de cifrar e uma chave $K$ . O **Decifrador** através da função caótica produz $\overline{K}$, o texto cifrado $\overline{C}$ e recupera o texto aberto $P$.

O criptossistema caótico apresentado na figura 3.2 possui um **cifrador** e um **decifrador** [YAN 97]:

O **cifrador** consiste em uma função caótica e um algoritmo de cifrar chamado $E$. A chave $K$ é utilizada para cifrar o texto aberto $P$. O $C$ é o texto cifrado que entra novamente no sistema caótico como retro-alimentação.

O **decifrador** através da função caótica produz $\overline{K}$ e o texto cifrado $\overline{C}$. O decifrador, através da chave $\overline{K}$, função caótica, e do texto cifrado $\overline{C}$, recupera o texto aberto $P$.

## 3.7 Cifradores que utilizam equações caóticas

Existem diversos criptossistemas que utilizam cifradores caóticos [ROS 99, GAL 96, SVE 96, INS 96, INS 98]. Na maioria dos casos, é utilizada a mesma função caótica que foi discutida na seção 3.2.6.

Alguns dos criptossistemas caóticos são utilizados para cifrar textos como descritos em [ROS 99, GAL 96, SVE 96] e outros criptossistemas descritos em [INS 96, INS 98] são adequados para cifrar imagens.

## 3.8 Cifradores de Roskin

Roskin e Caster desenvolveram um criptossistema utilizando equação caótica e o chamaram de cifrador simples. Para suprir as deficiências do cifrador simples e diminuir as vulnerabilidades existentes, criam o cifrador avançado.

### 3.8.1 Cifrador Simples

O primeiro criptossistema desenvolvido é denominado **cifrador simples** pelo fato de ter um cifrador de bloco simples com uma chave de 256 bits e bloco de 8 bits. Esse cifrador é ilustrado na figura 3.3. A seqüência de funcionamento desse cifrador, está descrita a seguir [ROS 99]:

1. A chave é usada para gerar uma função *pad* que é mesclada com o texto aberto em blocos de 8 bits (1 byte cada vez).

2. São utilizadas duas sucessivas chaves de seção ($k_i$ e $k_{i+1}$)

3. Estas chaves no lugar de serem adicionadas diretamente ao texto aberto, são utilizadas como **condições iniciais** para uma equação caótica.

4. A caixa $M_1$ representa um mapeamento de um byte (inteiros entre 0 e 255) para o domínio da equação caótica que são todos os reais no intervalo $[0, 1]$.

5. A soma da próxima chave de seção com o número 16 é utilizada como número de iterações da equação caótica.

6. A caixa $M_2$ mapeia todos os reais do domínio da equação caótica com intervalo $[0, 1)$, para um byte, com intervalo $[0, 255]$.

7. Ao cifrar cada novo bloco, $i$, o contador utilizado para manter a chave de seção atual é incrementado.

8. Os dados de saída da equação caótica são mesclados posteriormente com o texto aberto para obter texto cifrado.

9. Para decifrar, a mesma função *pad* é gerada e neste momento é retirada do texto cifrado, para obter o texto aberto.

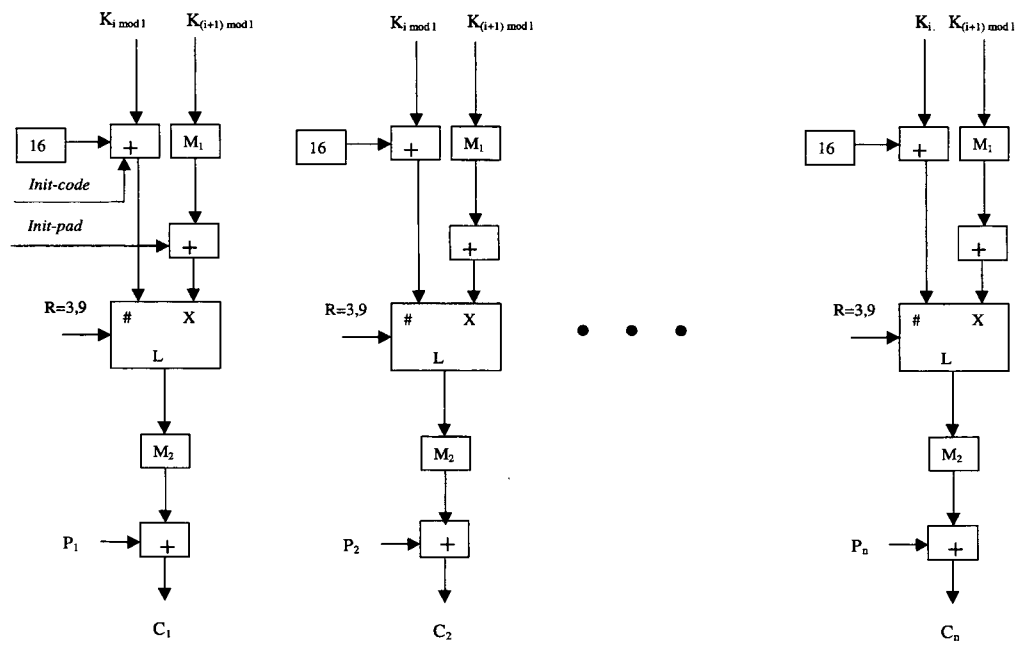


Figura 3.3: Cifrador Simples de Roskin

**Análise do Cifrador**

Os autores [ROS 99], analisaram e testaram este cifrador. Para saber como o cifrador funciona internamente e até que ponto é seguro, eles selecionaram imagens como no caso Torre Eifell para cifrar e decifrar.

Após a visualização de imagens cifradas e imagens planas, foi detectado um problema da função *pad* [ROS 99]:

O cifrador depende somente da chave, tem um período igual ao tamanho da chave e mostra comportamento periódico. Isto implica na utilização das mesmas séries de *pads* respectivamente, criação de modelos de deslocamento e aumenta a vulnerabilidade do criptossistema.

### 3.8.2 Cifrador Avançado

Para resolver o comportamento periódico do cifrador simples, foi desenvolvido um mecanismo de retro-alimentação e incorporado no cifrador, e assim foi criado o **Cifrador Avançado**, como visto na figura 3.4. O processo de cifrar de cada bloco do texto aberto não mais depende somente da chave, mas também do texto cifrado anterior. O *code*, que é bloco cifrado do texto aberto e *pad* gerado, são adicionados novamente ao sistema e servem como retro-alimentação do sistema.

Neste modelo, as séries de *pads* utilizadas dependem do texto cifrado e não apresentam comportamento periódico.

Para decifrar, o *pad* adicionado é retirado como mostra a figura 3.5.

## 3.9 Criptossistema Gao

Em 9 de dezembro de 1997, o Instituto Internacional de Ciência da Informação, de Tokyo, patenteou métodos para cifrar e decifrar informações, utilizando um sinal digital de Caos. Para cifrar o texto aberto, é gerado um sinal de caos que,

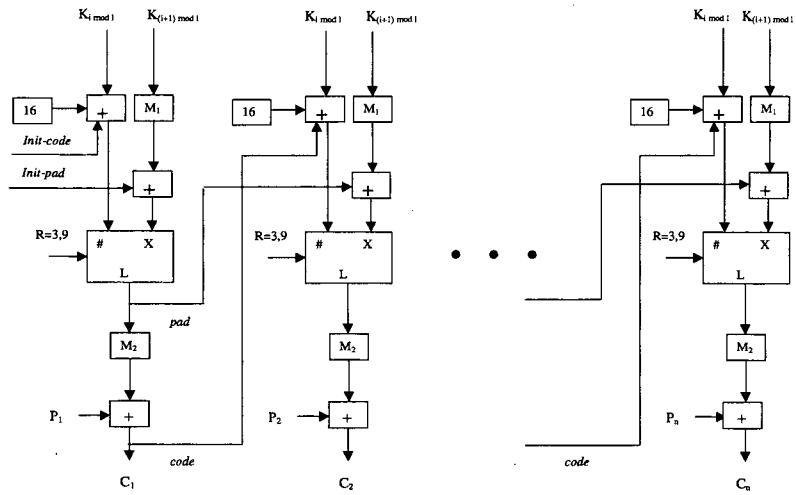


Figura 3.4: Esquema de cifrar do cifrador Avançado de Roskin. O processo de cifrar de cada parte do texto aberto não mais depende somente da chave, mas também do texto cifrado anterior.

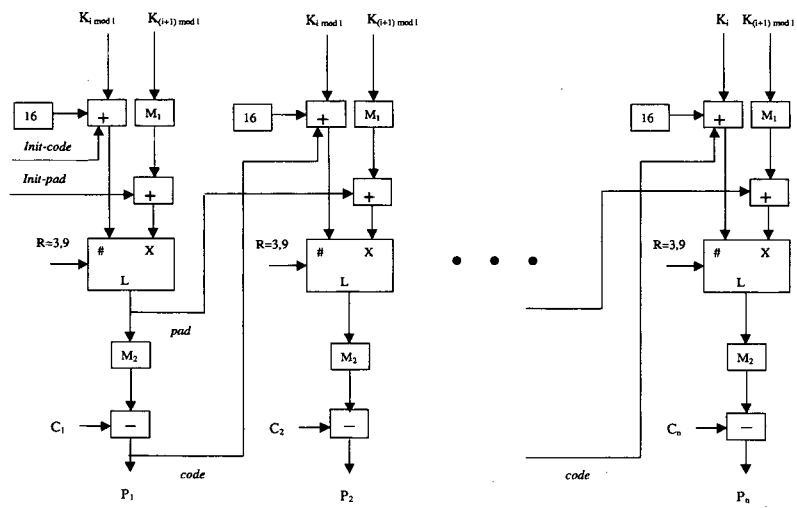


Figura 3.5: Esquema de decifrar do cifrador Avançado de Roskin. Para decifrar, basta retirar o *pad* adicionado.

junto com uma chave para cifrar, são adicionados ao texto aberto. Para recuperar o texto aberto, utiliza-se o mesmo sinal de caos gerado junto com a chave utilizada para cifrar. A figura 3.6[1] apresenta o criptossistema utilizado pelo **GCC**. (GCC é abreviatura de Gao's Chaos Cryptosystem.)

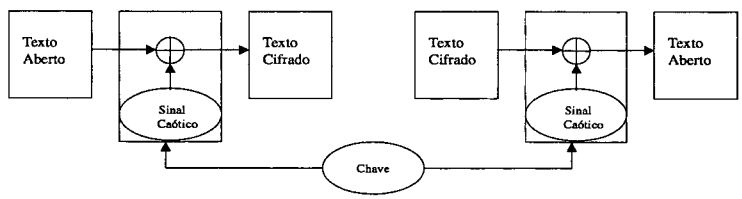


Figura 3.6: Criptossistema de GCC

Algumas características específicas do GCC são [INS 96, INS 98]:

1. É adequado para dados de multimídia (texto, som, gráfico, etc);
2. Atinge alta velocidade (64 Mbps);
3. É multi-plataforma (MS-DOS, Win3.1, Win95, NT, Unix, Linux);
4. É poderoso contra ruído (quatro vezes mais poderoso que os cifradores de bloco convencional).

## 3.10 Conclusão

O caos está presente em diversos campos da ciência. Os conceitos fundamentais da **Teoria do Caos** e as características dos sistemas caóticos foram detalhados neste capítulo. A aplicabilidade da **Teoria do Caos** no campo da criptografia também foi abordado neste capítulo. Foram caracterizados alguns criptossistemas que utilizam a **Teoria do Caos**, em especial os criptossistemas simples e avançado de Roskin que foram a base dos experimentos realizados nos capítulos 5 e 6, a seguir.

[1]Traduzida para português da figura que consta em [INS 96]

# Capítulo 4

# Criptossistemas Caóticos

## 4.1 Introdução

Este capítulo aborda os criptossistemas caóticos. Um criptossistema caótico caracteriza-se por ter um gerador de números randômicos que produz as condições iniciais do criptossistema. Por isso, as características de números pseudo-randômicos, números randômicos e seus respectivos geradores são estudados. Este capítulo verifica como deveria ser um criptossistema ideal, como também define o papel das funções caóticas nos criptossistemas, apresentando geradores de números pseudo-randômicos caóticos.

## 4.2 Números Pseudo-randômicos

Os números pseudo-randômicos são bastante utilizados em diversas aplicações de engenharia, bem como em sistemas de comunicação moderna [KOT 00]. Existem diversos algoritmos que geram números pseudo-randômicos.

Em criptografia, os números pseudo-randômicos são fundamentais. Os geradores de números pseudo-randômicos são utilizados para geração, distribuição de chaves de sessão e algoritmos de autenticação [STA 99].

Segundo [KOC 01], uma característica essencial que os números randô-

micos devem ter, é que sejam termos da seqüência de caracteres computacionalmente incompreensíveis e imprevisíveis, e que não apresentem nenhuma dependência entre eles. De forma intuitiva, esta seqüência é tratada como sendo **verdadeiramente randômica** e chamada de **algoritmicamente randômica**.

Ainda, segundo Kocarev [KOC 01], para analisar a qualidade de um número randômico, devem ser levadas em consideração a complexidade algorítmica e sua complexidade computacional.

Segundo Stallings em [STA 99], dois critérios são utilizados para verificar se uma seqüência de números pode ser considerada como randômica:

1. **Distribuição Uniforme**: A distribuição de termos em uma seqüência deve ser uniforme, de modo que a ocorrência de cada número seja aproximadamente a mesma.
2. **Independência**: Os termos da seqüência não devem possuir, nenhuma dependência entre si.

Kotulski, [KOT 00], cita que a qualidade dos algoritmos geradores de números randômicos é relacionada à dependência dos termos gerados. Quanto menor for a dependência entre os termos gerados, maiores serão a qualidade e a precisão do algoritmo e os termos gerados serão uniformemente distribuídos.

Segundo Kocarev [KOC 01], um subconjunto de uma seqüência de caracteres algoritmicamente randômica pode se tornar computacionalmente previsível. Infelizmente, é muito difícil decidir quando uma dada seqüência é randômica. Por isso, geralmente utilizamos seqüência de caracteres pseudo-randômicos.

A seqüência de termos pseudo-randômicos assemelha se com a seqüência de termos randômico, mas não pode ser considerada randômica. Esta seqüência é determinística e os termos desta seqüência têm dependência entre si: A partir de um subconjunto de termos desta seqüência, pode-se produzir as seqüências anteriores e posteriores da seqüência dada.

As linguagens de programação e os sistemas operacionais, entre outros aplicativos existentes, dispõem apenas de mecanismos para geração de seqüência de caracteres **pseudo-randômicos**.

## 4.3 Geradores de Números Pseudo-randômicos

Segundo Stallings em [STA 99], um dos algoritmos mais utilizados para geração de números pseudo-randômicos foi proposto, primeiramente, por Lehmer, o qual é conhecido também por **método congruencial linear**. A seqüência de números randômicos pode ser obtida através da seguinte equação:

$$X_{n+1} = (aX_n + c) mod\, m \tag{4.1}$$

Onde $m$ é o módulo para $m > 0$, $a$ é o multiplicador quando $0 \leq a < m$. O $c$ é o incremento no intervalo $0 \leq c < m$ e $X_0$ é o valor inicial no intervalo $0 \leq X_0 < m$.

Para valores inteiros de $m$, $a$, $c$ e $X_0$, poderemos produzir seqüência de números inteiros no intervalo $0 \leq X_n < m$.

Quanto maior for o $m$, maior será a seqüência de números randômicos gerados. Normalmente para o valor de $m$ utiliza-se o número $2^{31}$ que é o maior número inteiro positivo de um dado computador [STA 99].

Se o módulo $m$ e o multiplicador $a$ forem escolhidos de forma adequada, a seqüência de termos gerada é estatisticamente indistinguível com a seqüência de números randômicos. Entretanto, exceto o valor inicial $X_0$, que é escolhido de forma randômica, os outros números da seqüência são determinísticos [STA 99] e isto os torna previsíveis e vulneráveis.

## 4.4 Criptossistema Ideal

O esquema ideal para cifrar dados seria a utilização do algoritmo *One Time Pad* [STA 99]. O problema da utilização deste modelo é que o tamanho da chave utilizada deve ser igual ao tamanho do texto aberto. Com isto, quanto maior for o texto aberto, maiores serão o tempo e o espaço necessários para geração da chave e, conseqüentemente, haverá dificuldade de transporte da mesma.

Uma forma de resolver este problema é a utilização de geradores de

números randômicos, mas como visto na seção 4.2, os sistemas operacionais e linguagens de programação possuem apenas geradores de números pseudo-randômicos. Será que podem ser utilizados geradores pseudo-randômicos? Esta questão será analisada em seguida.

Seja $P$ um texto aberto, $R$ uma seqüência de números pseudo-randômicos e $C$ um texto cifrado temos a seguinte equação:

$$C_n = P_n \oplus R_n \tag{4.2}$$

onde:

$$P = \{P_0, P_1, P_2, ..., P_n\} \tag{4.3}$$

$$R = \{R_0, R_1, R_2, ..., R_n\} \tag{4.4}$$

$$C = \{C_0, C_1, C_2, ..., C_n\} \tag{4.5}$$

Supõe-se que um criptoanalista tenha conhecimento de $P_0$ e $C_0$, então:

$$C_0 = P_0 \oplus R_0 \tag{4.6}$$

Operando um **OU Exclusivo** com $P_0$ em ambos os lados desta equação obtem-s obtem-se:

$$P_0 \oplus C_0 = P_0 \oplus P_0 \oplus R_0 \tag{4.7}$$

$$P_0 \oplus C_0 = 0 \oplus R_0 \tag{4.8}$$

$$P_0 \oplus C_0 = R_0 \tag{4.9}$$

Assim, pode-se concluir que, com o conhecimento de $P_0$ e $C_0$, o criptoanalista consegue obter o $R_0$. Uma vez descoberto o $R_0$ e como este é um número pertencente a seqüência pseudo-randômica, podem ser gerados outros termos da seqüência e desta forma, o criptoanalista poderá alterar o texto aberto ou texto cifrado .

Então é necessário estabelecer uma forma de gerar $R_i$ que, dada $R_j$ qualquer, não se consiga obter $R_i$.

## 4.5 Geradores de Números Pseudo-randômicos Caóticos

Números caóticos possuem uma seqüência de números determinísticos que são considerados pseudo-randômicos devido a sua característica de **Dependência Sensitiva**.

Entre os geradores de números pseudo-randômicos amplamente utilizados em diversas aplicações, geradores de números pseudo-randômicos caóticos possuem propriedades atrativas, que garantem a unicidade da seqüência gerada para qualquer semente escolhida e independência entre os termos gerados desta seqüência [KOT 00].

Segundo ainda [KOT 00], os geradores de números pseudo-randômicos que são congruenciais lineares, descrito na seção 4.3 e demonstrado pela equação (4.1), são previsíveis e têm período finito. Os geradores de números randômicos caóticos têm períodos teoricamente infinitos.

Como a seqüência gerada neste caso possui comportamento randômico e não há dependência entre os termos gerados, se o criptoanalista obtiver uma única semente tal como $R_0$, não poderá desta, gerar seqüências anteriores e posteriores.

Acredita-se que uma forma de geração de números randômicos seja a produção de **números irracionais** que são caóticos. Como exemplo podem ser citados:

$$\sqrt{2}, \sqrt{3}, a\sqrt{2} + b\sqrt{3}, \sqrt{2+\sqrt{3}} \tag{4.10}$$

Ou as raízes de uma equação polinomial, desde que seja irracional:

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x + a_0 = 0 \tag{4.11}$$

Outro exemplo é:

$$ax^2 + bx + c = 0 \tag{4.12}$$

Segundo [GON 99], para valores irracionais de $\theta$, a equação (4.13) pode ser considerada como uma equação caótica.

$$X_n = \sin^2 (\theta \pi z^n) \tag{4.13}$$

Ainda em [GON 99], os autores afirmam que depois da análise rigorosa desta equação, para valores fracionários de $z > 1$, a equação (4.13) não é somente caótica, mas também é impossível calcular o próximo termo da equação, a partir dos termos anteriores. Quando $z$ não é inteiro, a condição inicial $X_0$ define unicamente o valor de $\theta$. Para valores fracionários de $z$ isto não acontece. Existem infinitos valores de $\theta$ que satisfazem as condições iniciais.

## 4.6 Conclusão

A utilização do algoritmo *One Time Pad* é desejável e seria o esquema ideal para cifrar o texto aberto. No *One Time Pad* é necessário que a chave utilizada seja do mesmo tamanho que o texto aberto e seja também randômica, o que dificultaria a sua utilização.

As ferramentas atuais de geração de números randômicos são, na verdade, geradores de números pseudo-randômicos, uma vez que é muito difícil a implementação de mecanismos geradores de seqüências **algoritmicamente randômicas**. Os números gerados com tais ferramentas são determinísticos e apresentam dependência entre si. Desta forma, descobrindo um único termo da seqüência, podem ser gerados outros termos da seqüência e assim, aumentar a vulnerabilidade do criptossistema.

Entre os mecanismos geradores de números pseudo-randômicos deve-se escolher aquele que verdadeiramente apresente o comportamento randômico e que tenha pouca ou nenhuma dependência entre os termos.

Os algoritmos lineares, como o algoritmo descrito na seção 4.2, são previsíveis e os termos da seqüência gerada são determinísticos.

Para diminuir a previsibilidade dos algoritmos lineares, foi citada, como uma solução, a utilização de geradores de números pseudo-randômicos caóticos. Como por exemplo, raízes de uma equação polinomial.

# Capítulo 5

# Avaliação de Criptossistemas Caóticos

## 5.1 Introdução

Para descobrir a qualidade dos criptossistemas caóticos e a sua vulnerabilidade, é necessário avaliá-los. Para poder avaliar tais criptossistemas é preciso propor métodos de avaliação. Uma forma de avaliação é através da comparação de suas características com as características desejáveis dos criptossistemas.

Por esta razão, este capítulo compara as características de criptossistemas caóticos com as características desejáveis dos criptossistemas atuais, a fim de propor um método que avalie os criptossistemas caóticos.

A segurança de um criptossistema está relacionada com a sua não linearidade. Neste capítulo descreve-se os experimentos realizados com o criptossistema avançado de Roskin, em relação a compactação de dados e o Critério da Avalanche Estrita.

## 5.2 Características desejáveis dos criptossistemas

Nesta seção, baseado Stallings [STA 99], descreve-se as principais características de avaliação para criptossistemas simétricos:

1. *Tamanho da chave variável*

   A variação do tamanho da chave de um criptossistema contribui de forma significativa na sua segurança. Quanto maior for o tamanho da chave utilizada, menor será a chance de decifrar o texto cifrado por criptoanálise.

2. *Operadores mistos*

   Os operadores mistos complicam a descoberta do texto cifrado e contribuem para não linearidade do criptossistema.

3. *Rotação dependente do dado*

   A rotação dependente do dado cria confusão e difusão de dados.

4. *Rotação dependente da chave*

   A rotação dependente da chave também aumenta segurança do criptossistema.

5. *Tamanho do texto aberto e texto cifrado variável*

   A variação do tamanho do bloco tanto para texto aberto, como para texto cifrado, é uma estratégia interessante para criptossistemas e contribui no aumento da segurança do criptossistema.

6. *Número de rodadas variável*

   A variação do número de rodadas em um criptossistema também dificulta o trabalho de Criptoanálise.

Estas características podem ser utilizadas para avaliação dos criptossistemas. A tabela 5.1 mostra uma avaliação do criptossistema proposto por Roskin [ROS 99] que foi descrito na seção 3.8.

## 5.3 Avaliação de Criptossistemas Caóticos

A figura 5.1 mostra o esquema típico de um criptossistema. Se o gerador de números randômicos for uma função caótica, pode-se considerar este esquema, como

Tabela 5.1: Avaliação do cifrador de Roskin

| **Características** | **Avaliação** |
|---|---|
| Tamanho da Chave Variável | Atende, pois permite criar chaves de até 256 bytes |
| Operadores Mistos | Não possui |
| Rotação dependente do dado | Não possui |
| Rotação dependente da chave | Não possui |
| Tamanho do texto aberto e texto cifrado variável | Não possui |
| Número de rodadas variável | Não possui |

um esquema de um criptossistema caótico. Através da operação **OU EXCLUSIVO** entre o texto aberto $P$ e a seqüência randômica gerada pelo Gerador de Números Randômicos, é criado o texto cifrado $C$.

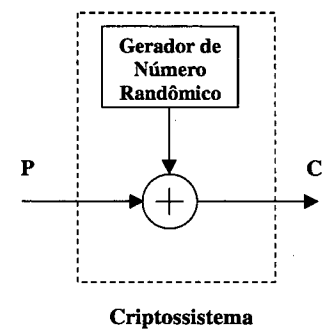


Figura 5.1: Esquema típico de um criptossistema. Se o gerador de números randômicos for uma função caótica, pode-se considerar este esquema, como um esquema de um criptossistema caótico.

Pela dificuldade inerente dos criptossistemas caóticos, na formalização do criptossistema com vistas a sua análise, como pode-se avaliar a segurança destes criptossistemas?

Para esta avaliação, existem duas possíveis formas:

1. O gerador de números randômicos deve gerar números verdadeiramente randômicos

que não tenham dependência entre si e que não seja possível obter o termo $X_{n+1}$ a partir de qualquer um dos termos anteriores, como foi visto na seção 4.3.

2. Avaliar o comportamento do sistema como um todo. A segurança de um criptossistema está relacionada à sua não-linearidade. Existem inúmeras formas de se avaliar esta não-linearidade. Dentre as possíveis formas, podem ser utilizadas as seguintes técnicas:

   (a) Observar a não-linearidade através da compactação de dados.

   (b) Verificar o Critério da Avalanche Estrita (Strict Avalanche Criterion).

## 5.4 Observar a não linearidade através da compactação de dados

A compactação de dados utiliza diversas técnicas para minimizar redundância de dados e reuní-los em conjuntos menores. Uma taxa elevada de compactação indica alta redundância existente no arquivo de dados. Quanto maior for a taxa de compactação de um arquivo, maior será a vulnerabilidade à criptoanálise do texto cifrado.

Segundo Schneier, [SCH 98], se tentar compactar um texto cifrado e a taxa de compactação for pequena ou zero, é bom indicativo para concluir que o criptossistema utilizado tem boa qualidade, no sentido que provavelmente existem poucas redundâncias do texto cifrado que poderiam facilitar a criptoanálise.

### 5.4.1 Definição dos modelos

Para observar a não-linearidade através da técnica de compactação de dados, inicialmente foram criados dez modelos que serviram como textos abertos, para que os experimentos realizados tenham ampla variedade de dados de entrada. A tabela 5.2 mostra os modelos criados.

Tabela 5.2: Modelos utilizados nos experimentos de compactação. Para que os experimentos realizados tenham ampla variedade de dados de entrada, foram criados dez modelos.

| **Modelo** | **Descrição** |
|---|---|
| 01 | texto aberto com bit 1, no formato seqüencial **11111111....** |
| 02 | texto aberto com bit 0, no formato seqüencial **00000000....** |
| 03 | texto aberto com metade de bits de 1 na primeira parte e metade de bits de 0 na segunda parte. |
| 04 | texto aberto com metade de bits de 0 na primeira parte e metade de bits de 1 na segunda parte. |
| 05 | texto aberto com metade de bits de 0 e metade de bits de 1 no formato **01010101..1**. |
| 06 | texto aberto com metade de bits de 1 e metade de bits de 0 no formato **10101010..0**. |
| 07 | texto aberto com formato seqüencial **01110111...**, começando com **0**. |
| 08 | texto aberto com formato seqüencial **10001000...**, começando com **1**. |
| 09 | texto aberto com formato seqüencial **11101110...**, começando com **1**. |
| 10 | texto aberto com formato seqüencial **00010001...**, começando com **0**. |

### 5.4.2 Etapas necessárias

Para verificar a não linearidade através da compactação de dados seguintes etapas foram feitas:

1. Gerar os Arquivos Modelos que serão os **textos abertos**.
2. Cifrar os textos abertos, utilizando o cifrador do Roskin.
3. Cifrar os textos abertos, utilizando um cifrador qualquer.
4. Compactar os textos abertos gerados, utilizando um compactador.
5. Compactar os textos cifrados por cifrador do Roskin.
6. Compactar os textos cifrados por um cifrador qualquer.
7. Comparar as taxas de compactação entre os textos abertos e cifrados.

As etapas 1 a 6 foram detalhadas no apêndice B e o código fonte do gerador de modelos encontra-se no apêndice B.6.

### 5.4.3 Comparação das taxas de compactação

Após ter concluído as etapas de cifrar e compactar, pode-se comparar as taxas de compactação obtida em cada uma. A figura 5.2 mostra este processo. O texto aberto $P$ que foi gerado em diversos formatos, como descrito na tabela 5.2, é cifrado com cifrador $AES$ e o cifrador de $Roskin$. Estes cifradores produzem os textos cifrados $C_1$ e $C_2$. Os textos cifrados são compactados com o compactador `winzip`[1] e geram os textos compactados $C'$ e $C"$, respectivamente. Logo depois é feita a comparação das taxas obtidas de compactação por cada cifrador.

A tabela 5.3 mostra as taxas de compactação que foram obtidas para textos abertos de 1024 bits. Como pode ser observado na tabela, os textos abertos de 1024

[1] http://www.winzip.com

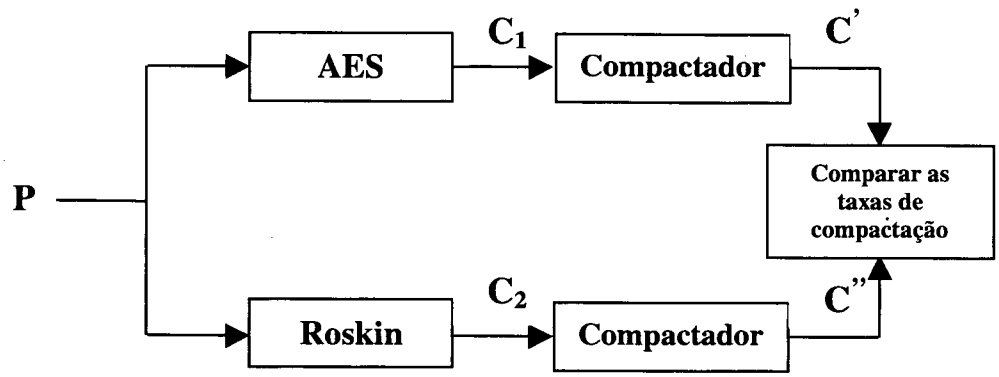


Figura 5.2: Comparação das taxas de compactação. O texto aberto $P$ é cifrado com o cifrador $AES$ e o cifrador $Roskin$. Os textos cifrados resultantes são compactados com $winzip$ e as taxas de compactação obtidas são comparadas.

bits apresentam taxa de compactação de 95% em média, enquanto os textos cifrados com cifrador de Roskin e cifrador AES, do mesmo tamanho, apresentam taxa de compactação de 0%.

Para obter melhores resultados, foram repetidas estas etapas para textos abertos de 1k bytes, 4k bytes, 8k bytes, 32k bytes, 64k bytes, 128k bytes, 256k bytes, 512k bytes, 1M byte, 2M bytes, 4M bytes, 8M bytes e 32M bytes.

Os textos abertos de 1k apresentam taxa de compactação de 99% em média, enquanto os textos cifrados com cifrador de Roskin e cifrador AES, do mesmo tamanho, apresentam taxa de compactação de 0%, conforme mostrado na tabela 5.4.

Os textos abertos de 4k bytes apresentam taxa de compactação de 100% em média, enquanto os mesmos textos quando cifrados com cifrador de Roskin, apresentam taxa de 5% em média, e quando cifrados com cifrador AES, permanecem com a taxa de compactação de 0%, conforme a tabela 5.5.

### 5.4.4 Análise dos resultados obtidos

A tabela 5.6 ilustra as taxas obtidas entre diversos modelos de 128 bytes a 32M bytes. Analisando os dados da tabela 5.6, pode-se concluir que os textos abertos gerados, pelo gerador de modelos, variando de 128 bytes a 32M bytes, apresentam taxa

Tabela 5.3: Taxas de Compactação para textos abertos de 1024 bits. A taxa de compactação obtida para textos abertos neste caso é de 95% em média.

| Modelo | Taxa de compactação do texto aberto (%) | Taxa de compactação do Texto Cifrado com Cifrador Roskin(%) | Taxa de compactação do Texto Cifrado com Cifrador AES(%) |
|---|---|---|---|
| 01 | 95 | 0 | 0 |
| 02 | 95 | 0 | 0 |
| 03 | 93 | 0 | 0 |
| 04 | 93 | 0 | 0 |
| 05 | 95 | 0 | 0 |
| 06 | 95 | 0 | 0 |
| 07 | 95 | 0 | 0 |
| 08 | 95 | 0 | 0 |
| 09 | 95 | 0 | 0 |
| 10 | 95 | 0 | 0 |
| Média | 95 | 0 | 0 |

Tabela 5.4: Taxas de Compactação para textos abertos de 1k byte. A taxa de compactação obtida para textos abertos neste caso é de 99% em média.

| Modelo | Taxa de compactação do texto aberto (%) | Taxa de compactação do Texto Cifrado com Cifrador Roskin(%) | Taxa de compactação do Texto Cifrado com Cifrador AES(%) |
|---|---|---|---|
| 01 | 99 | 0 | 0 |
| 02 | 99 | 0 | 0 |
| 03 | 99 | 0 | 0 |
| 04 | 99 | 0 | 0 |
| 05 | 99 | 0 | 0 |
| 06 | 99 | 0 | 0 |
| 07 | 99 | 0 | 0 |
| 08 | 99 | 0 | 0 |
| 09 | 99 | 0 | 0 |
| 10 | 99 | 0 | 0 |
| Média | 99 | 0 | 0 |

Tabela 5.5: Taxas de Compactação para textos abertos de 4k bytes. A taxa de compactação obtida com textos abertos neste caso é de 100% em média.

| Modelo | Taxa de compactação do texto aberto (%) | Taxa de compactação do Texto Cifrado com Cifrador Roskin(%) | Taxa de compactação do Texto Cifrado com Cifrador AES(%) |
|---|---|---|---|
| 01 | 100 | 5 | 0 |
| 02 | 100 | 5 | 0 |
| 03 | 100 | 5 | 0 |
| 04 | 100 | 5 | 0 |
| 05 | 100 | 5 | 0 |
| 06 | 100 | 5 | 0 |
| 07 | 100 | 5 | 0 |
| 08 | 100 | 5 | 0 |
| 09 | 100 | 5 | 0 |
| 10 | 100 | 5 | 0 |
| Média | 100 | 5 | 0 |

de compactação de 100%, em média. Os mesmos dados quando são cifrados pelo cifrador de Roskin apresentam taxa de 6% de compactação, em média. Porém, os mesmos textos abertos, quando cifrados por cifrador de AES, permanecem com a taxa de compactação de 0%. Conforme os dados obtidos, o cifrador de Roskin, pelo critério da compactação, é mais vulnerável do que o cifrador AES.

Tabela 5.6: Taxas de Compactação para diversos modelos. As taxas de compactação obtidas com cifrador de Roskin são superiores das taxas obtidas com cifrador AES. Isto indica que o cifrador de Roskin é mais vulnerável do que o cifrador AES.

| Modelo (bytes) | Taxa de compactação do texto aberto (%) | Taxa de compactação do Texto Cifrado com Cifrador Roskin(%) | Taxa de compactação do Texto Cifrado com Cifrador AES(%) |
|---|---|---|---|
| 128 | 95 | 0 | 0 |
| 1k | 100 | 0 | 0 |
| 4k | 100 | 5 | 0 |
| 8k | 100 | 6 | 0 |
| 32k | 100 | 6 | 0 |
| 64k | 100 | 6 | 0 |
| 128k | 100 | 6 | 0 |
| 256k | 100 | 6 | 0 |
| 512k | 100 | 6 | 0 |
| 1M | 100 | 6 | 0 |
| 2M | 100 | 6 | 0 |
| 4M | 100 | 6 | 0 |
| 8M | 100 | 6 | 0 |
| 32M | 99 | 6 | 0 |
| Média | 100 | 6 | 0 |

## 5.5 Verificar o Critério da Avalanche Estrita (Strict Avalanche Criterion)

Na seção 2.7.2, foi definido o Critério de Avalanche Estrita como uma propriedade desejável para os criptossistemas. Esta seção apresenta uma forma de analisar este critério.

### 5.5.1 Como verificar o Critério da Avalanche Estrita?

A quantidade de possibilidades para arquivos que têm um bit de diferença pode ser resultado da equação (5.1), baseado no trabalho de Darú [DAR 01]. Seja $P$ , número de possibilidades para um arquivo de $n$ bits, temos:

$$P = n2^{n-1} \tag{5.1}$$

Para concluir se um dado criptossistema atende ao Critério da Avalanche Estrita (SAC), deve-se verificar, se **todas** as alterações de um único bit existente para o **texto aberto** e a **chave**, resultam em alteração de **metade** de bits no texto cifrado. A seguir ilustra-se este conceito através de alguns exemplos:

1. **Arquivo de 2 bits**

   Para um arquivo de 2 bits, existem as seqüências de **00, 01, 10 e 11**. Para cada seqüência deve-se analisar quais outras seqüências têm somente um bit de diferença. A tabela 5.7 mostra as diversas possibilidades para cada seqüência, onde ocorre alteração de um bit, para um arquivo de 2 bits, totalizando **4** possibilidades.

2. **Arquivo de 3 bits**

   Para um arquivo de 3 bits, as seqüências possíveis são: **000, 001, 010, 011, 100, 101, 110 e 111**. As possibilidades de alteração de um bit para um arquivo de 3 bits são **12**. A tabela 5.8 ilustra as diversas seqüências, onde são alteradas apenas um bit.

Tabela 5.7: Troca de um bit para um arquivo de 2 bits

| **Seqüência 00** | **Seqüência 01** | **Seqüência 10** |
|---|---|---|
| 01 | 11 | 11 |
| 10 | | |

Tabela 5.8: Troca de um bit para um arquivo de 3 bits

| **Seqüência 000** | **Seqüência 111** | **Seqüência 001** | **Seqüência 010** | **Seqüência 101** |
|---|---|---|---|---|
| 001 | 011 | 011 | 011 | 101 |
| 010 | 101 | 101 | 110 | 110 |
| 100 | 110 | | | |

A próxima seção, verifica se o criptossistema de Roskin atende ao Critério da Avalanche Estrita.

### 5.5.2 Criptossistema de Roskin atende ao SAC?

Para verificar se o criptossistema Roskin atende ao SAC, deve-se verificar a variação do texto cifrado quando o texto aberto varia, mantendo a chave constante. Em seguida, deve-se variar a chave, mantendo o texto aberto constante (que por convenção tem o mesmo valor que o arquivo de chave) e verificar a variação do texto cifrado.

Para esta finalidade, utiliza-se o programa SAC para cada número específico de bits, que foi desenvolvido em C e está no apêndice B. Este apêndice também contém os resultados dos experimentos realizados com o criptossistema de Roskin, utilizando modelos de 2 a 15 bits referente ao SAC.

Para os experimentos realizados, foram obtidos o desvio padrão através da equação (5.2).

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(\overline{x} - x_i)^2}{n-1}} \tag{5.2}$$

Onde $\sigma$ = Desvio Padrão, n = número de amostras, $\overline{x}$ é média aritmética e $x_i$ é valor da amostra.

### 5.5.3 Análise dos resultados obtidos

Conforme os experimentos realizados com o criptossistema de Roskin, descritos no apêndice B, pode-se concluir que com a variação de apenas **um bit** do texto aberto, existe uma grande variação no texto cifrado. Isto resulta em **Efeito Avalanche**.

Nos modelos de 8 a 15 bits, com a variação de apenas **um bit** da chave, a variação no texto cifrado apresenta uma simetria com modelo gaussiano, onde se aproxima bastante do **Critério da Avalanche Estrita**.

Para que um criptossistema atenda ao SAC ou se aproxime ao SAC, é necessário que, com a variação de um bit do texto aberto ou da chave, a variação do texto cifrado apresente um modelo gaussiano.

Desta forma, o criptossistema de Roskin, conforme experimentos detalhados no apêndice B, não atende ao SAC e apenas apresenta um **Efeito Avalanche**.

## 5.6 Conclusão

Este capítulo apresentou duas formas de avaliação de não-linearidade. A primeira forma é a compactação de dados que permite descobrir a taxa de redundância de de um dado texto. Quanto menor for a taxa obtida no processo de compactação de um dado texto, conseqüentemente este, será menos vulnerável aos ataques e tentativas de criptoanalista.

Os experimentos de compactação com o criptossistema de Roskin e o cifrador AES foram apresentados neste capítulo. Este capítulo também comparou as taxas de compactação obtidas através dos diversos modelos gerados, com o criptossistema de Roskin e com o cifrador AES.

Segunda forma de avaliação de não-linearidade é através do **Critério da Avalanche Estrita**. Se a alteração de apenas **um bit** do texto aberto ou da chave, em um criptossistema, resulta exatamente em troca de metade dos bits do texto cifrado, pode-se afirmar que tal criptossistema atende ao SAC.

Baseado nos experimentos realizados com relação ao **Efeito da Avalanche Estrita** para os modelos de 2 a 15 bits apresentados neste capítulo, pode-se concluir que com a variação de apenas **um bit** do texto aberto, existe uma grande variação no texto cifrado. Isto resulta em **Efeito Avalanche**.

Entretanto, variando um bit da chave, nos modelos de 2 a 15 bits, a variação do texto cifrado apresenta uma simetria com modelo gaussiano, onde se aproxima bastante do **Critério da Avalanche Estrita**, mas na realidade é apenas o **Efeito Avalanche**.

# Capítulo 6

# Modificações no Criptossistema de Roskin

## 6.1 Introdução

Conforme os experimentos realizados na seção anterior, concluiu-se que o criptossistema de Roskin não atende o Critério da Avalanche Estrita e apresenta taxas maiores de compactação comparadas com o cifrador AES.

O objetivo deste capítulo é propor alterações com objetivo de melhorar o criptossistema de Roskin e verificar se as melhorias propostas auxiliam na redução das vulnerabilidades existentes no criptossistema de Roskin. Para esta finalidade foram implementados três modelos.

O primeiro modelo utiliza esquema único para cifrar e decifrar, retirando o esquema de retro-alimentação do criptossistema de Roskin e implementando o operador **OU EXCLUSIVO**. Inserindo texto aberto $P$ no criptossistema, resulta em texto cifrado $C$. Se inserir o texto cifra $C$ , pode-se obter o texto aberto $P$. Como este modelo não utiliza o esquema de retro-alimentação, torna-se mais vulnerável, não foram realizados experimentos referentes a compactação e o Efeito da Avalanche Estrita.

O segundo modelo implementa o operador **OU EXCLUSIVO**, já que o cifrador original do Roskin não o tinha. Este modelo é simplesmente chamado de modelo

XOR.

O terceiro modelo, que ao longo deste capítulo é chamado de modelo reverso, inverte o texto cifrado e o insere no criptossistema novamente como um texto aberto. Isto é feito para aumentar o efeito de confusão e difusão dos dados. Além disso, neste modelo também é utilizado o operador **OU EXCLUSIVO** que o cifrador de Roskin não possui.

Para concluir-se qual dos dois últimos modelos apresenta desempenho melhor, foram realizados novamente os experimentos relativos a compactação e SAC, conforme descrito no capítulo anterior.

## 6.2 Esquema único para cifrar e decifrar

Utilizando operador **OU-EXCLUSIVO**, e tirando a retro-alimentação do **code**[1], pode-se criar um único esquema para cifrar e decifrar, como a figura 6.1 mostra. Conforme a figura, **A** pode ser um texto **aberto** ou **cifrado** e conseqüentemente **B** será **cifrado** ou **aberto**.

Este modelo apresenta um único esquema para cifrar e decifrar, mas como não utiliza o esquema de retro-alimentação, torna-se mais vulnerável.

## 6.3 Criptossistema XOR

No primeiro modelo, chamado simplesmente de criptossistema XOR, foi implementado o operador de **OU EXCLUSIVO (XOR)** que o criptossistema de Roskin não possui. Para avaliar este modelo, foram realizados os experimentos referentes a **Compactação de Dados** e **SAC**.

A figura 6.2 mostra o esquema de cifrar do criptossistema XOR. O cifrador do criptossistema XOR utiliza o operador de **OU EXCLUSIVO**, quando o texto aberto $P_1$ é introduzido no cifrador.

[1]Código alterado se encontra no apêndice D

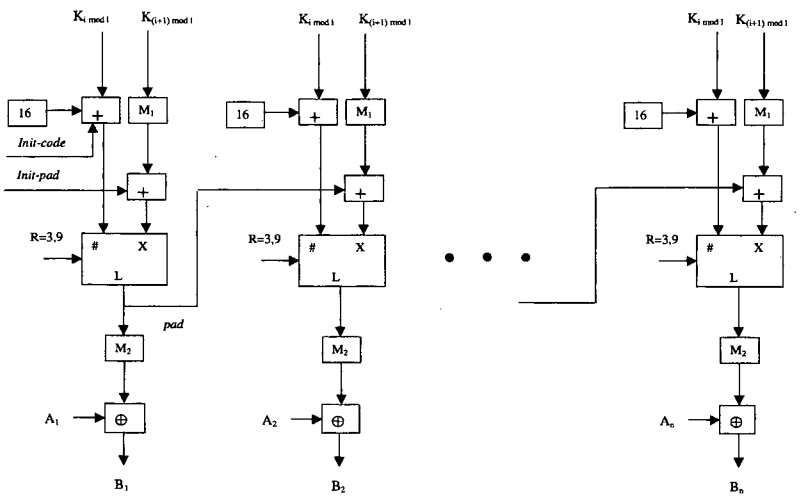


Figura 6.1: Esquema único de cifrar e decifrar para o criptossistema de Roskin

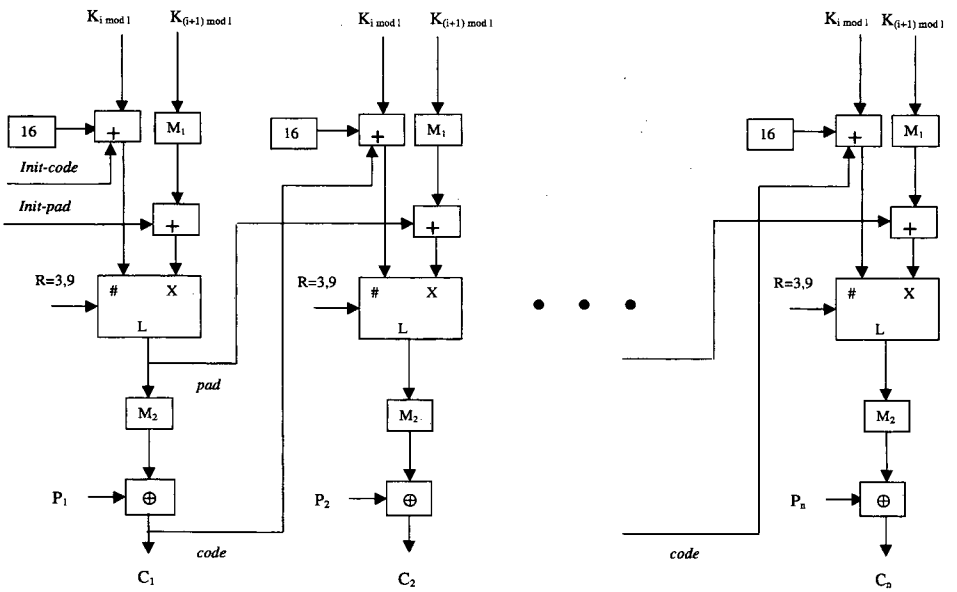


Figura 6.2: Cifrador do criptossistema XOR

Na figura 6.3 pode-se verificar o decifrador do criptossistema XOR. O decifrador do criptossistema XOR utiliza o operador de **OU EXCLUSIVO**, quando o texto cifrado $C_1$ é introduzido no decifrador.

Para poder avaliar este criptossistema, foram realizados os experimentos de **compactação** e verificado novamente o Critério da Avalanche Estrita.

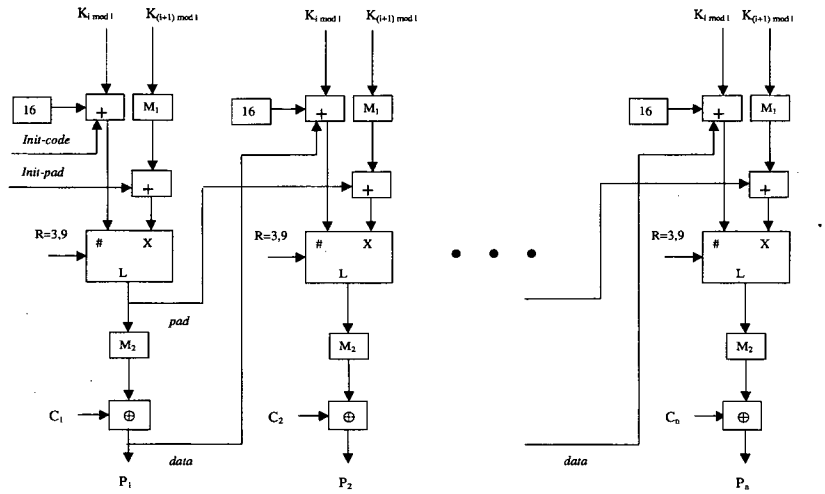


Figura 6.3: Decifrador do criptossistema XOR

### 6.3.1 Compactação de dados com Criptossistema XOR

Nesta seção foram utilizados os modelos propostos no apêndice B.2.

1. Utilizar os mesmos textos abertos que foram criados no apêndice B.2.
2. Cifrar os textos abertos, utilizando o criptossistema XOR.
3. Compactar os textos abertos.
4. Comparar as taxas de compactação entre os textos abertos e os textos cifrados.

**Cifrar os textos abertos, utilizando o criptossistema XOR**

Para cifrar os textos abertos utilizando sistema operacional DOS, foram utilizados os seguintes passos:

1. **for i in *.txt**: Para todos os arquivos com extensão **txt**.
2. **do**: Faça
3. **criptoxo chave %i n%i**: cifra os textos abertos, gerando os textos cifrados que começam com **n**.

**Compactar os textos cifrados com criptossistema XOR**

Foi utilizado o mesmo utilitário (**winzip**) para compactação de textos cifrados.

**Comparar as taxas de compactação**

Após ter concluído as etapas de cifrar e compactar, pode-se comparar as taxas de compactação obtida em cada etapa. A tabela 6.1 mostra as taxas de compactação que foram obtidas para diversos tipos de arquivos. As taxas de compactação obtidas com o modelo XOR, em média são de 6%. Esta média indica que existem indícios de redundância de dados cifrados com o modelo XOR. Este modelo não reduziu a redundância existente no criptossistema de Roskin.

### 6.3.2 Criptossistema XOR atende SAC?

Para esta finalidade, foi utilizado programa SAC, para modelos de 2 a 15 bits, que foi desenvolvido em C e está no apêndice B. Os detalhes dos modelos, os gráficos e os resultados obtidos referente Critério da Avalanche Estrita estão no apêndice E.1.

### 6.3.3 Análise dos resultados obtidos

Conforme os resultados obtidos nesta seção, pode-se concluir que o modelo XOR não atende ao Efeito da Avalanche Estrita. Com a variação da chave, o modelo se aproxima bastante a uma curva gaussiana, enquanto com a variação do texto aberto, isto não ocorre.

Comprando estes resultados com os resultados obtidos no criptossistema de Roskin, conclui-se que o modelo XOR não contribui no aumento de difusão e confusão de dados e conseqüentemente, não reduz as vulnerabilidades existentes no criptossistema de Roskin.

Tabela 6.1: Taxas de Compactação obtidas com o modelo XOR

| Modelo | Taxa de compressão do Texto Cifrado com o modelo XOR(%) |
|---|---|
| 128 bytes | 0 |
| 1k byte | 0 |
| 4k bytes | 5 |
| 8k bytes | 6 |
| 32k bytes | 6 |
| 64k bytes | 6 |
| 128k bytes | 6 |
| 256k bytes | 6 |
| 512k bytes | 6 |
| 1M byte | 6 |
| 2M bytes | 6 |
| 4M bytes | 6 |
| 8M bytes | 6 |
| 32M bytes | 6 |
| Média | 6 |

## 6.4 Criptossistema Reverso

Esta seção apresenta o esquema do criptossistema reverso. O texto cifrado resultante do criptossistema de Roskin será gravado no formato inverso e introduzido no sistema somente uma vez, como um texto aberto. O objetivo do criptossistema reverso é aumentar a **Confusão** e **Difusão** de dados. A figura 6.4 ajuda a entender o funcionamento deste modelo. O texto aberto $P_1$ quando cifrado, produz o texto cifrado $C_1'$. O texto cifrado é invertido e introduzido no cifrador novamente como um texto aberto, $P_n'$. O $P_n'$ é cifrado novamente e é produzido o texto cifrado $C_n$.

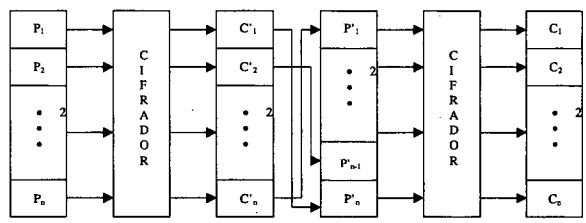


Figura 6.4: Cifrador do criptossistema Reverso

A figura 6.5 ilustra o esquema de cifrar através do criptossistema reverso, cujo código fonte encontra-se no apêndice D. O modelo reverso utiliza o operador **OU EXCLUSIVO** e foi baseado no criptossistema de Roskin.

A figura 6.6, ilustra o processo do decifrador, cujo código fonte encontra-se no apêndice D. Para decifrar, o texto cifrado $C_1$ é introduzido no decifrador e é obtido o texto aberto $P_1$. O $P_1$ é invertido e introduzido novamente no decifrador como $C_n$, e deste é obtido o texto aberto $P_n$.

Para avaliar este modelo, foram realizados os experimentos de **compactação** e do Critério da Avalanche Estrita.

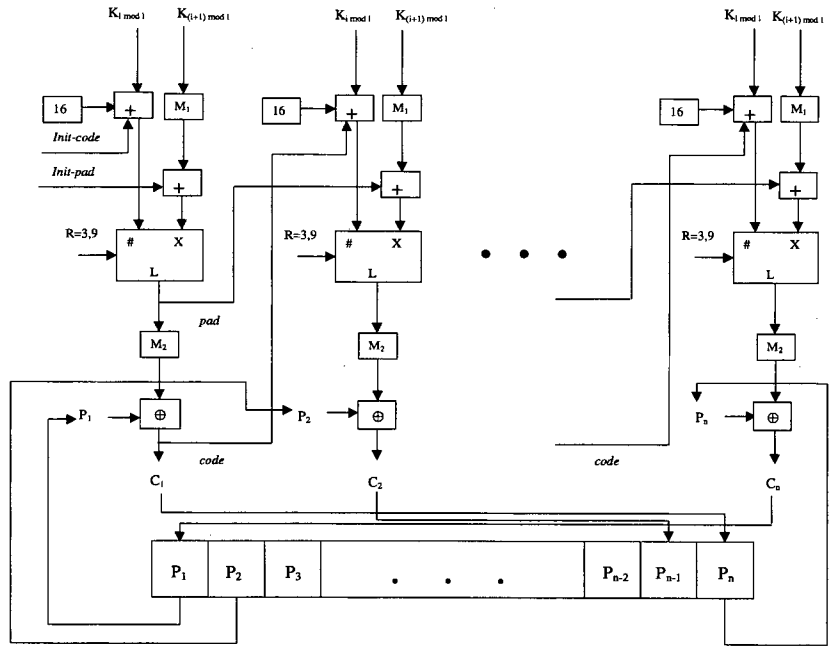


Figura 6.5: Cifrador do Criptossistema Reverso

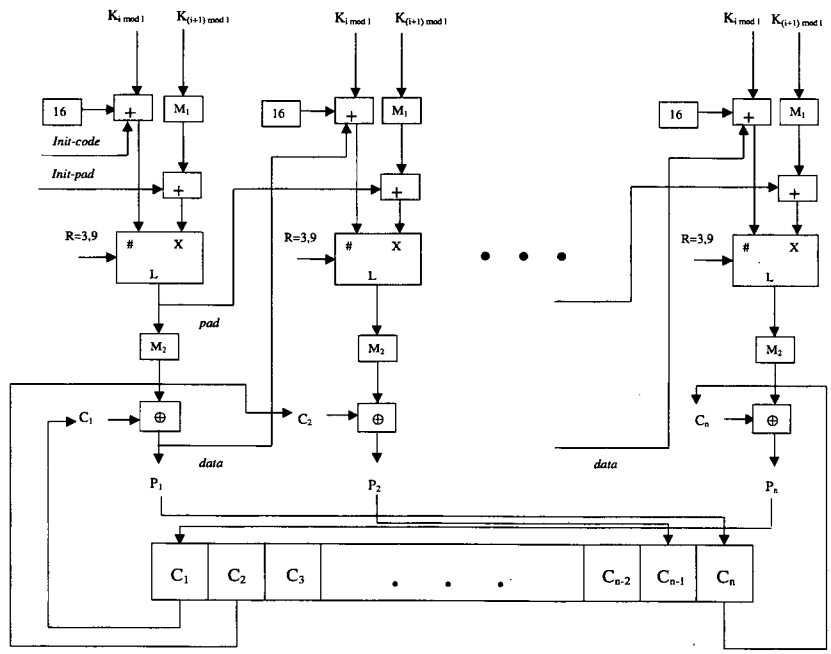


Figura 6.6: Decifrador do criptossistema Reverso

### 6.4.1 Compactação de dados com cifrador Reverso

Nesta seção foram utilizados os modelos propostos no apêndice B.2. Para atender esta parte foram feitos seguintes passos para arquivos de 128 bytes até 32M:

1. Utilizar os mesmos textos abertos que foram criados no apêndice B.2.
2. Cifrar os textos abertos, utilizando o cifrador Reverso.
3. Compactar os textos abertos gerados.
4. Comparar as taxas de compactação entre os textos abertos e os textos cifrados.

**Cifrar os textos abertos, utilizando o cifrador Reverso**

Para cifrar os textos abertos, no sistema operacional DOS, foi utilizado o seguinte *batch* de comandos:

1. **for i in *.txt**: Para todos os arquivos com extensão **txt**.
2. **do**: Faça
3. **criptore %i chave r%i**: gera os textos cifrados que começam com **r**.

**Compactar os textos cifrados com o cifrador Reverso**

Para esta finalidade foi utilizado o compactador **winzip**.

**Comparar as taxas de compactação**

Após ter concluídas as etapas de cifrar e compactar, pode-se comparar as taxas de compactação obtida em cada etapa. A tabela 6.2 mostra as taxas de compactação que foram obtidas para diversos modelos com diversos tamanhos. Os dados obtidos mostram que há pouco indício de redundância de dados, nos dados cifrados com este cifrador. Conseqüentemente, isto contribui no aumento de segurança do cifrador reverso.

Tabela 6.2: Taxas de Compactação obtidas com o cifrador reverso

| Modelo | Taxa de compressão do Texto Cifrado com o Cifrador Reverso(%) |
|---|---|
| 128 bytes | 0 |
| 1k byte | 0 |
| 4k bytes | 0 |
| 8k bytes | 0 |
| 32k bytes | 0 |
| 64k bytes | 0 |
| 128k bytes | 0 |
| 256k bytes | 0 |
| 512k bytes | 0 |
| 1M byte | 0 |
| 2M bytes | 0 |
| 4M bytes | 0 |
| 8M bytes | 0 |
| 32M bytes | 0 |
| Média | 0 |

### 6.4.2 Criptossistema Reverso atende SAC?

Para poder verificar se o criptossistema reverso atende o critério SAC, foram feitos os mesmos experimentos já realizados com o criptossistema de Roskin, utilizando o programa **SAC**, que encontra-se no apêndice B.

Os detalhes dos modelos, os gráficos e os resultados obtidos referente Critério da Avalanche Estrita estão no apêndice E.2.

### 6.4.3 Análise dos resultados obtidos

Conforme os resultados obtidos e detalhados na seção 6.4, observa-se a aproximação do modelo reverso a uma curva gaussiana, nos modelos com maior número de bits. Esta aproximação, variando o texto aberto ou a chave, começa a ocorrer para os modelos de 8 a 15 bits.

Baseado nestes dados, apesar do modelo reverso não atender completamente ao Efeito da Avalanche Estrita, este se aproxima bastante deste modelo. Esta aproximação contribui significativamente no efeito de difusão e confusão de dados e conseqüentemente na redução das vulnerabilidades existentes no criptossistema de Roskin.

## 6.5 Conclusão

Nesta capítulo foram propostos três modelos, dos quais dois modelos foram testados e implementados. Para validar até que ponto os modelos propostos contribuem para a redução das vulnerabilidades existentes no criptossistema de Roskin e qual destes apresenta melhor desempenho em termos de compactação e Efeito da Avalanche Estrita, foram realizados novamente as etapas de compactação de dados e a validação do Critério da Avalanche Estrita.

Analisando os dados obtidos com o cifrador reverso e cifrador proposto, pode-se concluir que, no cifrador proposto, a diferença entre os dados analíticos quando varia-se o texto aberto ou a chave é pequena. No entanto, no cifrador proposto, existe uma diferença maior entre os dados analíticos quando varia-se o texto aberto ou a chave.

O modelo reverso apesar de não atender completamente ao Efeito da Avalanche Estrita, está muito próximo pela sua distribuição gaussiana de dados. Esta aproximação contribui na redução das vulnerabilidades existentes do criptossistema de Roskin.

Como foi visto, o cifrador proposto não apresenta nenhuma diferença com o criptossistema de Roskin, em termos de compactação. Entretanto, o cifrador reverso, que apresenta a mesma taxa de compactação do cifrador AES, diminui a redundância de dados e aumenta a segurança do criptossistema de Roskin.

# Capítulo 7

# Considerações Finais

A utilização de equações caóticas nos cifradores com objetivo de aumentar a segurança já faz parte do cotidiano, e alguns exemplos foram vistos neste documento. O que falta para estes criptossistemas é o emprego de uma metodologia para avaliação dos mesmos.

O objetivo deste trabalho de dissertação, foi desenvolver uma metodologia que auxilie na avaliação dos criptossistemas caóticos, propor algumas modificações no criptossistema de Roskin e avaliar tais modificações.

Os fundamentos de criptografia foram apresentados no capítulo 2. Os princípios da teoria de caos foram apresentados no capítulo 3. O capítulo 3 apresentou e estudou os criptossistemas caóticos. Os geradores de números randômicos e os geradores de números randômicos caóticos foram detalhados no capítulo 4. Foram realizados diversos experimentos com o criptossistema de Roskin e descritos no capítulo 5 e no apêndice B. O capítulo 5 apresentou métodos para avaliação dos criptossistemas caóticos. A avaliação do criptossistema de Roskin através da técnica de Compressão de Dados o e Critério de Avalanche Estrita, foi apresentado no capítulo 5. Foram propostas algumas modificações no criptossistema de Roskin, propondo novos criptossistemas que foram apresentados no capítulo 6.

## 7.1 Objetivos Alcançados

Os objetivos propostos no capítulo 1 foram alcançados, exceto o último, que foi propor novos geradores de números randômicos caóticos. O estudo realizado mostra que esta seria uma tarefa demasiadamente complexa.

Como trabalho futuro sugere-se o estudo dos geradores de números randômicos caóticos que foi visto no capítulo 4.2. Pois, acredita-se que os números verdadeiramente randômicos, contribuem significativamente para o aumento da segurança de criptossistemas caóticos. A descoberta da relação entre os termos de uma equação polinomial de grau $n$, da forma que apresente no mínimo uma raíz irracional, também contribui para formar uma seqüência de números randômicos. Em que condição e até que ponto, um gerador de números pseudo-randômicos caótico pode ser imprevisível é problema central de **Criptografia Baseada em Sistemas Caóticos** [KOC 01].

## 7.2 Análise dos resultados obtidos

Para o processo de avaliação de criptossistemas caóticos, foram utilizadas duas técnicas, e feitos experimentos descritos ao longo dos capítulos deste trabalho.

Com a primeira técnica, através da **compactação de dados**, descobre-se o grau da redundância de um dado texto. Quanto menor for esta redundância, mais difícil se torna a compactação do texto. Então, quanto menor a taxa obtida no processo de compactação, há menos dados redundantes no texto e, conseqüentemente este, será menos vulnerável aos ataques e tentativas do criptoanalista.

Os experimentos realizados com cifrador de Roskin utilizando esta técnica, foram apresentados no capítulo 5 e no apêndice B. Através dos resultados obtidos e ilustrados nas tabelas contidas nos tais capítulos, conclui-se que o criptossistema de Roskin apresenta taxas maiores de compactação se comparado com cifrador AES. Por isso, foram propostos dois modelos de melhoria e os experimentos realizados foram repetidos com tais modelos e apresentados no capítulo 6. Através dos resultados obtidos, pode-se concluir que o primeiro modelo proposto, simplesmente chamado de XOR, apresenta

taxas maiores de compactação, quando comparado com o cifrador AES. Portanto, este modelo não contribui na redução das vulnerabilidades existentes no criptossistema de Roskin, já que esta taxa indica indício de redundância nos dados cifrados. O segundo modelo proposto, simplesmente chamado de reverso, tem a mesma taxa de compactação quando comparado com cifrador AES. Isto quer dizer que o dado cifrado pelos cifradores propostos não é compactável, pois a taxa de compactação é de zero porcento. Assim, este modelo reverso contribui significativamente na redução das vulnerabilidades existentes no criptossistema de Roskin.

Através da segunda técnica, **Efeito da Avalanche Estrita**, ou simplesmente chamado por **SAC**[1], verifica-se os aspectos de confusão e difusão do texto que são características desejáveis de criptossistemas e diminuem a vulnerabilidade dos criptossistemas, como também aumentam a segurança.

Os experimentos realizados com esta técnica, com cifrador de Roskin foram apresentados no capítulo 5 e no apêndice B. Baseado nos resultados obtidos e através das diversas figuras e tabelas estatísticas nos tais capítulos, pode-se concluir que para os modelos de até 15 bits, o criptossistema de Roskin se aproxima ao SAC, quando varia-se o arquivo de chave. Mas variando o texto plano, não há tal aproximação e, na verdade, o criptossistema de Roskin produz apenas um Efeito Avalanche no texto, variando texto plano ou variando arquivo de chave.

Os mesmos experimentos foram realizados com os modelos de melhoria propostos e apresentados no capítulo 6. Com referência aos resultados obtidos, através das diversas figuras e tabelas estatísticas neste capítulo, conclui-se que para os modelos até 15 bits, o segundo modelo proposto, simplesmente chamado reverso, a partir de 10 bits, começa a se aproximar ao SAC, variando o texto plano ou arquivo chave. Enquanto o primeiro modelo, apenas se aproxima ao SAC, variando o arquivo de chave. Quando varia-se o texto plano, este modelo apenas apresenta um Efeito Avalanche.

Com os resultados obtidos nas duas técnicas utilizadas, pode-se concluir que o segundo modelo proposto, modelo reverso, contribui na redução das vulnerabilidades existentes no criptossistema de Roskin, através da redução da redundância de dados

[1] SAC é abreviatura de Strict Avalanche Criterion

cifrados e também apresenta confusão e difusão de dados, através do efeito de Avalanche Estrita.

Com a intenção de ser referência no campo de criptografia caótica, esta dissertação poderá ser ponto de partida para novos trabalhos de pesquisa que proponham técnicas para avaliação de criptossistemas e a geração de novos criptossistemas caóticos.

# Bibliografia


[BAP 98] BAPTISTA, M. S. Cryptography with chaos. **Physics Letters A**, Maryland, v.240, p.50 – 54, 1998.

[CAL 00] CALOYANNIDES, M. A. Encryption wars: Early battles. **IEEE Spectrum**, [S.l.], v.37, n.4, p.37–43, April, 2000.

[DAR 01] DARÚ, G. H. Trabalho sobre probabiliade, curva normal e ajuste de curvas. Trabaho de disciplina de Métodos Matemáticos, Junho, 2001.

[dC 00] DE CARVALHO, D. B. **Segurança de Dados Com Criptografia**. RJ: Editora Book Express Ltda, 2000.

[DRA 94] DRAZIN, P. G. **Nonlinear Systems**. New York: Cambridge University Press, 1994.

[FER 94] FERRARA, N. F.; PRADO, C. P. C. **Caos Uma Introdução**. São Paulo: Editora Edgard Blücher Ltda, 1994.

[FOR 00] FORD, S. Advanced encryption standard (aes) questions and answers. `http://www.nist.gov/`, October, 2000.

[GAL 96] GALLAGHER, J. B.; GOLDSTEIN, J. Sensitive dependence cryptography. `http://www.navigo.com/sdc`, 1996.

[GLE 00] GLEICK, J. **Caos A Criação de uma Nova Ciência**. São Paulo: Editora Campus Ltda, 2000.

[GON 99] GONZáLEZ, J. A.; PINO, R. A random number generator based on unpredictable chaotic functions. **Computer Physics Communications**, Caracas, v.120, p.109 –114, 1999.

[HO 95] HO, A. Chaos introduction. `http://www.zeuscat.com/andrew/chaos/chaos.html`, November, 1995.

[INS 96] INSTITUTE, I. I. S. Gao's chaos cryptosystem overview. `http://www.iisi.co.jp/reserch/GCC-over.htm`, October, 1996.

[INS 98] INSTITUTE, I. I. S. Cryptograph for multimedia age - gcc chaos cryptosystem. `http://www.iisi.co.jp/reserch/`, April, 1998.

[KIM 91] KIM, K. Construction of des-like s-boxes based os boolean functions satisfying the sac. In: ASIACRYPT, 1991. **Proceedings...** Japan: Springer, 1991. v.739, p.59–72.

[KOC 01] KOCAREV, L. Chaos and cryptography. 2001. **Proceedings...** USA: [s.n.], 2001.

[KOT 00] KOTULSKI, Z.; SZCZEPANSKI, J. On constructive approch to chaotic pseudorandom number generators. In: NATO REGIONAL CONFERENCE ON MILITARY COMMUNICATIONS AND INFORMATION SYSTEMS, 2000. **Proceedings...** Poland: [s.n.], 2000. p.191–203.

[LEE 99] LEE, A. Guideline for implementing cryptography in the federal government. `http://www.nist.gov`, November, 1999.

[LEN 99] LENSTRA, A. K.; VERHEUL, E. R. Selecting cryptography key sizes. November, 1999.

[LOP 00] LOPES, F. A fortaleza dos negócios. **Network Computing**, [S.l.], v.2, p.30–36, Fevereiro, 2000.

[MAS 88] MASSEY, J. L. An introduction to contemporary cryptology. 1988. **Proceedings...** Zurich: [s.n.], 1988. v.76, p.533–548.

[MEI 00] MEISS, J. D. Nonlinear science faq, version 1.4, May, 2000. `http://www.faqs.org/faqs/sci/nonlinear-faq/index.html`.

[MEN 96] MENEZES, A.; OORSCHOT, P. V.; VANSTONE, S. **Handbook of Applied Cryptography**. California: CRC Press, 1996.

[RAE 98] RAE, G. Chaos theory: A brief introduction. `http://www.imho.com/grae/chaos/chaos.html`, January, 1998.

[RDS 98] RSA DATA SECURTIY, I. RSA labratories frequently asked questions about today's cryptography, v4.0, 1998. `http://www.rsa.com`.

[ROS 96] ROSS, M.; TRAEGER, M.; KRAYNAK, A. The chaos experience. `http://library.thinkquest.org/3120/text/math.htm`, November, 1996.

[ROS 99] ROSKIN, K. M.; CASTER, J. B. From chaos to cryptography. `http://xcrypt.theory.org`, March, 1999.

[SCH 98] SCHNEIER, B. **Applied Cryptography**. second. ed. New York: John Wiley & Sons, Inc., 1998.

[SEB 94] SEBERRY, J.; ZHANG, X.-M.; ZHENG, Y. Improving the strict avalanche characteristics of cryptographic functions. **Information Processing Letters**, Australia, v.50, p.37 – 41, 1994.

[STA 99] STALLINGS, W. **Cryptography and Network Security Principles and Practice**. 2nd. ed. New Jersey: Prentice Hall, 1999.

[SVE 96] SVENSSON, M.; MALMQUIST, J.-E. A simple secure communications system utilizing chaotic funtions to control the encryption and decryption of messages. Lund Institute of Technology, 1996. Relatório técnico.

[TER 00] TERADA, R. **Segurança de Dados - Criptografia em Redes de Computador**. SP: Editora Edgard Blücher Ltda, 2000.

[TRU 98] TRUMP, M. A. What is chaos? `http://order.ph.utexas.edu`, November, 1998.

[WU 99] WU, L. Discrete-time chaotic encryption system: Design and analysis. `http://security.ece.orst.edu/koc/ece575/5/99Project`, March, 1999.

[YAN 97] YANG, T.; WAH, C.; CHUA, L. O. Cryptography based on chaotic systems. In: IEEE TRANSCANCTIONS ON CIRCUITS AND SYSTEMS, 1997. **Proceedings...** Berkeley: [s.n.], 1997. v.44, p.469 – 471.

[YAO 98] YAO, K. Fundamental issues in chaotic communication systems. 1998. **Proceedings...** USA: [s.n.], 1998.

# Apêndice A

# Códigos fontes dos criptossistemas de Roskin

Este apêndice contém os códigos fonte do criptossistema do Roskin [ROS 99] que foram mencionados na sessão 3.8.

## A.1 Cifrador Simples de Roskin

```
#include <iostream.h>
#include <fstream.h>
void load_session_keys(ifstream& key_file,
unsigned char session_keys[ ]);
double init_pad(unsigned char session_keys[ ]);
unsigned char init_code(unsigned char session_keys[ ]);
double logistic(double x, int iterations);
int next_subkey(int key);
double map_to(unsigned char x);
unsigned char map_from(double x);
double chop(double x);

const double r = 3.9;

int main(int argc, char *argv[ ]) {
    if(argc != 3) {
        cout << "Usage: xcrypt key_file plaintext_file" << endl;
        return 10;
    }

    ifstream key_file(argv[1]);
    if(!key_file.is_open()) {
        cerr << "Error: Can't open key file." << endl;
        return 10;
    }
    unsigned char session_keys[32];
    load_session_keys(key_file, session_keys);
    key_file.close();

    ifstream plaintext(argv[2]);
    if(!plaintext.is_open()) {
        cerr << "Error: Can't plaintext file." << endl;
```

```
        return 10;
    }
    double pad = init_pad(session_keys);
    unsigned char code = init_code(session_keys);
    unsigned char data;
    int subkey = 0;
    int c;
    while( (c = plaintext.get()) != EOF) {
        data = c;
        pad = logistic(    chop(map_to(session_keys[subkey])),
                       16 + session_keys[next_subkey(subkey)]
                       );
        code = (int(data) + map_from(pad)) % 256;
        cout << char(code);
        subkey = next_subkey(subkey);
    }
    plaintext.close();
    return 0;
}

void load_session_keys(ifstream& key_file,
unsigned char session_keys[ ])
{
    for(int i = 0; i < 32; i++)
        key_file >> session_keys[i];
}

double init_pad(unsigned char session_keys[ ]) {
    unsigned char pad = session_keys[0];
    for(int i = 1; i < 32; i++)
        pad = pad ^ session_keys[i];
    return double(pad) / 256.0;
}

unsigned char init_code(unsigned char session_keys[ ]) {
    unsigned char code = session_keys[0];
    for(int i = 1; i < 32; i++)
        code = (int(code) + int(session_keys[i])) % 256;
    return code;
}

double logistic(double x, int iterations) {
    for(int i = 0; i < iterations; i++)
        x = r*x*(1 - x);
    return x;
}

int next_subkey(int key) {
    return (key + 1 ) % 32;
}

double map_to(unsigned char x) {
    return double(x) / 256.0;
}

unsigned char map_from(double x) {
    return (unsigned char)(x * 256.0);
}
double chop(double x) {
    return x - int(x);
}
```

## A.2 Decifrador Simples de Roskin

```
#include <iostream.h>
#include <fstream.h>

void load_session_keys(ifstream& key_file,
unsigned char session_keys[ ]);
double init_pad(unsigned char session_keys[ ]);
unsigned char init_code(unsigned char session_keys[ ]);
double logistic(double x, int iterations);
int next_subkey(int key);
double map_to(unsigned char x);
unsigned char map_from(double x);
double chop(double x);

const double r = 3.9;

int main(int argc, char *argv[ ]) {
    if(argc != 3) {
        cout << "Usage: xdecrypt key_file cyphertext_file" << endl;
        return 10;
    }

    ifstream key_file(argv[1]);
    if(!key_file.is_open()) {
        cerr << "Error: Can't open key file." << endl;
        return 10;
    }
    unsigned char session_keys[32];
    load_session_keys(key_file, session_keys);
    key_file.close();

    ifstream cyphertext(argv[2]);
    if(!cyphertext.is_open()) {
        cerr << "Error: Can't cyphertext file." << endl;
        return 10;
    }

    double pad = init_pad(session_keys);
    unsigned char code = init_code(session_keys);

    unsigned char data;
    int subkey = 0;
    int c;

    while( (c = cyphertext.get()) != EOF) {
        data = c;
        pad = logistic(    chop(map_to(session_keys[subkey])),
                        16 + session_keys[next_subkey(subkey)]
                        );

        code = (int(data) - map_from(pad)) % 256;
        cout << char(code);

        code = data;
        subkey = next_subkey(subkey);
    }

    cyphertext.close();

    return 0;
}

void load_session_keys(ifstream& key_file,
unsigned char session_keys[ ])
{
    for(int i = 0; i < 32; i++)
        key_file >> session_keys[i];
}

double init_pad(unsigned char session_keys[ ]) {
    unsigned char pad = session_keys[0];
    for(int i = 1; i < 32; i++)
        pad = pad ^ session_keys[i];

    return double(pad) / 256.0;
}
```

```
unsigned char init_code(unsigned char session_keys[ ]) {
    unsigned char code = session_keys[0];
    for(int i = 1; i < 32; i++)
        code = (int(code) + int(session_keys[i])) % 256;
    return code;
}

double logistic(double x, int iterations) {
    for(int i = 0; i < iterations; i++)
        x = r*x*(1 - x);
    return x;
}

int next_subkey(int key) {
    return (key + 1 ) % 32;
}

double map_to(unsigned char x) {
    return double(x) / 256.0;
}

unsigned char map_from(double x) {
    return (unsigned char)(x * 256.0);
}

double chop(double x) {
    return x - int(x);
}
```

## A.3 Cifrador Avançado de Roskin

```
#include <iostream.h>
#include <fstream.h>

void load_session_keys(ifstream& key_file,
unsigned char session_keys[ ]);
double init_pad(unsigned char session_keys[ ]);
unsigned char init_code(unsigned char session_keys[ ]);
double logistic(double x, int iterations);
int next_subkey(int key);
double map_to(unsigned char x);
unsigned char map_from(double x);
double chop(double x);

const double r = 3.9;

int main(int argc, char *argv[ ]) {
    if(argc != 3) {
        cout << "Usage: xcrypt key_file plaintext_file" << endl;
        return 10;
    }

    ifstream key_file(argv[1]);
    if(!key_file.is_open()) {
        cerr << "Error: Can't open key file." << endl;
        return 10;
    }
    unsigned char session_keys[32];
    load_session_keys(key_file, session_keys);
    key_file.close();

    ifstream plaintext(argv[2]);
    if(!plaintext.is_open()) {
        cerr << "Error: Can't plaintext file." << endl;
        return 10;
    }

    double pad = init_pad(session_keys);
    unsigned char code = init_code(session_keys);

    unsigned char data;
    int subkey = 0;
    int c;

    while( (c = plaintext.get()) != EOF) {
        data = c;
        pad = logistic(    chop(map_to(session_keys[subkey]) + pad),
                        16 + session_keys[next_subkey(subkey)] + code
                        );

        code = (int(data) + map_from(pad)) % 256;
        cout << char(code);

        subkey = next_subkey(subkey);
    }

    plaintext.close();

    return 0;
}

void load_session_keys(ifstream& key_file,
unsigned char session_keys[ ])
{
    for(int i = 0; i < 32; i++)
        key_file >> session_keys[i];
}

double init_pad(unsigned char session_keys[ ]) {
    unsigned char pad = session_keys[0];
    for(int i = 1; i < 32; i++)
        pad = pad ^ session_keys[i];

    return double(pad) / 256.0;
}

unsigned char init_code(unsigned char session_keys[ ]) {
    unsigned char code = session_keys[0];
```

```
    for(int i = 1; i < 32; i++)
        code = (int(code) + int(session_keys[i])) % 256;
    return code;
}
double logistic(double x, int iterations) {
    for(int i = 0; i < iterations; i++)
        x = r*x*(1 - x);
    return x;
}
int next_subkey(int key) {
    return (key + 1 ) % 32;
}
double map_to(unsigned char x) {
    return double(x) / 256.0;
}
unsigned char map_from(double x) {
    return (unsigned char)(x * 256.0);
}
double chop(double x) {
    return x - int(x);
}
```

## A.4 Decifrador Avançado de Roskin

```
#include <iostream.h>
#include <fstream.h>

void load_session_keys(ifstream& key_file,
unsigned char session_keys[ ]);
double init_pad(unsigned char session_keys[ ]);
unsigned char init_code(unsigned char session_keys[ ]);
double logistic(double x, int iterations);
int next_subkey(int key);
double map_to(unsigned char x);
unsigned char map_from(double x);
double chop(double x);

const double r = 3.9;

int main(int argc, char *argv[ ]) {
    if(argc != 3) {
        cout << "Usage: xdecrypt key_file cyphertext_file" << endl;
        return 10;
    }

    ifstream key_file(argv[1]);
    if(!key_file.is_open()) {
        cerr << "Error: Can't open key file." << endl;
        return 10;
    }
    unsigned char session_keys[32];
    load_session_keys(key_file, session_keys);
    key_file.close();

    ifstream cyphertext(argv[2]);
    if(!cyphertext.is_open()) {
        cerr << "Error: Can't cyphertext file." << endl;
        return 10;
    }

    double pad = init_pad(session_keys);
    unsigned char code = init_code(session_keys);

    unsigned char data;
    int subkey = 0;
    int c;

    while( (c = cyphertext.get()) != EOF) {
        data = c;
        pad = logistic(    chop(map_to(session_keys[subkey]) + pad),
                        16 + session_keys[next_subkey(subkey)] + code
                        );

        code = (int(data) - map_from(pad)) % 256;
        cout << char(code);

        code = data;
        subkey = next_subkey(subkey);
    }

    cyphertext.close();

    return 0;
}

void load_session_keys(ifstream& key_file,
unsigned char session_keys[ ])
{
    for(int i = 0; i < 32; i++)
        key_file >> session_keys[i];
}

double init_pad(unsigned char session_keys[ ]) {
    unsigned char pad = session_keys[0];
    for(int i = 1; i < 32; i++)
        pad = pad ^ session_keys[i];

    return double(pad) / 256.0;
}
```

```
unsigned char init_code(unsigned char session_keys[ ]) {
    unsigned char code = session_keys[0];
    for(int i = 1; i < 32; i++)
        code = (int(code) + int(session_keys[i])) % 256;
    return code;
}
double logistic(double x, int iterations) {
    for(int i = 0; i < iterations; i++)
        x = r*x*(1 - x);
    return x;
}
int next_subkey(int key) {
    return (key + 1 ) % 32;
}
double map_to(unsigned char x) {
    return double(x) / 256.0;
}
unsigned char map_from(double x) {
    return (unsigned char)(x * 256.0);
}
double chop(double x) {
    return x - int(x);
}
```

# Apêndice B

# Experimentos Realizados

Este apêndice contém os códigos fonte e as diversas tabelas de entrada referente aos programas que foram utilizados para experimentos de compactação e Efeito de Avalanche Estrita.

## B.1 Experimentos de Compactação

Os passos a seguir descreve a seqüência de experimentos realizados com a compactação de dados:

1. Gerar os Arquivos Modelos que serão os **Textos Planos**.
2. Cifrar os textos planos, utilizando o cifrador do Roskin.
3. Cifrar os textos planos, utilizando um cifrador qualquer.
4. Compactar os textos planos gerados, utilizando um compactador.
5. Compactar os textos cifrados por cifrador do Roskin.
6. Compactar os textos cifrados por um cifrador qualquer.
7. Comparar as taxas de compactação entre os arquivos planos e cifrados.

## B.2 Geração dos Arquivos Modelos

São gerados os textos planos que servirão para os experimentos. Os textos planos gerados nesta etapa são de **1024** bits. Para esta finalidade, utiliza-se o programa **Gerador** cujo o código fonte encontra-se em anexo.

O arquivo **entrada.txt** contém as especificações para criação de modelos e as suas variações. A seguir pode-se conferir conteúdo do arquivo **entrada.txt** para modelos com **1024 bits** em tabela B.1.

Então com o arquivo **entrada.txt** os modelos anteriormente apresentados com 1024 bits cada um, são criados.

Tabela B.1: Arquivo de entrada para 1024 bits

| | |
|---|---|
| 10 | São gerados **10** Modelos |
| 1024 | Os arquivos gerados têm **1024** bits |
| 1 1 1024 | 1024 bits de 1, 111...1 |
| 1 0 1024 | 1024 bits de 0, 000...0 |
| 2 1 512 0 512 | 512 bits de 1 e 512 bits de 0, **1111...0000** |
| 2 0 512 1 512 | 512 bits de 0 e 512 bits de 1, **0000...1111** |
| 1 01 512 | 512 bits de **01**, **0101...01** |
| 1 10 512 | 512 bits de **10**, **1010...10** |
| 1 0111 256 | 256 seqüências de **0111** |
| 1 1000 256 | 256 seqüências de **1000** |
| 1 1110 256 | 256 seqüências de **1110** |
| 1 0001 256 | 256 seqüências de **0001** |

## B.3 Cifrar os textos planos, utilizando o cifrador do Roskin

Para cifrar os textos planos, foi utilizado seguinte comando no operacional DOS:

1. **for i in *.txt**: Para todos os arquivos com extensão **txt**.
2. **do**: Faça
3. **criptoca_file %i ¿ c%i**: cifra os arquivos, gerando os arquivos cifrados que começam com **c**.

## B.4 Cifrar os textos planos, utilizando um cifrador qualquer

Nesta seção os textos planos gerados são cifrados, com uma implementação do algoritmo **Rijndael** que foi escolhido pelo NIST, cujo código fonte se encontra em anexo.

Esta implementação é para ambiente **DOS**. Para poder cifrar os arquivos planos utiliza-se seguinte comando:

**for %i in (*.txt) do rijn-ofb %i¡key_file**

## B.5 Compactar os arquivos planos gerados

Para compactar os arquivos foi utilizado programa **winzip.exe**.

## B.6 Gerador de Textos Abertos

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include "bit.h"
void Altera_Modelo(char *, char *, int );
int main() {
  FILE *entrada, *saida;
  char *strModelo;
  char *strAux;
  char strPadrao[128],nome_arq[30];
  int iNumRep, iNumBits, iNumPadroes, iNumModelos;
  int i,j,k,iPos,bit;
  //Abre arquivo de modelos
  if( (entrada=fopen("entrada.txt","r"))==NULL)
  {
    printf("Erro de Abertura do arquivo entrada.txt");
//    getch();
    return 1;
  }
  //carrega o numero de modelos
  fscanf(entrada,"%i",&iNumModelos);
  //avanca ate proxima posicao de leitura
  while(fgetc(entrada)!='\n');
  //le numero de bits
  fscanf(entrada,"%i",&iNumBits);
  //Aloca memoria para armazenar os modelos
  strModelo = (char *) malloc( iNumBits/8 );
  //inicializa strModelo com todos os bits iguais a 0
  strAux = strModelo;
  //avanca ate proxima posicao de leitura
  while(fgetc(entrada)!='\n');
  for(i=0;i<iNumBits/8;i++)
    *(strAux++) = 0;
  //Laco externo para leitura de todos os modelos dos arquivos
  for(i=0;i<iNumModelos;i++)
  {
    //Le numero de padroes
    fscanf(entrada,"%d",&iNumPadroes);
    //Inicializa apontador de posicao a ser alterada
    iPos=0;
    //para cada padrao monta o padrao de bits
    for(j=0; j<iNumPadroes;j++)
    {
      //Le a string padrao
      fscanf(entrada,"%s",strPadrao);
      //Le o numero de repeticoes
      fscanf(entrada,"%i",&iNumRep);
      for(k=0;k<iNumRep;k++)
      {
Altera_Modelo(strModelo,strPadrao,iPos);
iPos+=strlen(strPadrao);
      } //Fim do Padrao
    }//Fim do Modelo
    //Escreve Modelo
    sprintf(nome_arq,"./MODELOS/%03dMod.txt",i);
    if( (saida=fopen(nome_arq,"w+")) == NULL )
      printf("Erro de abertura do arquivo: %s",nome_arq);
    else
    {
      fwrite(strModelo,iNumBits/8,1,saida);
      fclose(saida);
    }
/*    //Escreve variacoes
    //aloca memoria para strAux
    strAux = (char *) malloc( iNumBits/8 );
    for(j=iNumBits-1;j>=0;j--) {
      strcpy(strAux,strModelo);
      bit = bit_get(strAux,j);
      bit = bit==0? 1 : 0;
```

```
      bit_set(strAux, j ,bit);
      //Grava variacao
      sprintf(nome_arq,"./MODELOS/%07d.txt",j);
      if( (saida=fopen(nome_arq,"w+")) == NULL )
printf("Erro de abertura do arquivo: %s",nome_arq);
      else
      {
fwrite(strAux,iNumBits/8,1,saida);
fclose(saida);
      }
    }//Fim da escrita das variacoes
    //libera memoria de strAux
    free(strAux);*/
  }//Fim dos modelos
  fclose(entrada);
  return 0;
}

void Altera_Modelo(char *strModelo, char *strPadrao, int iPos)
{
  int i;
  for(i=0;i<strlen(strPadrao);i++)
    bit_set(strModelo,iPos+i,strPadrao[i]-'0');
}
```

## B.7 Programa para verificar o Efeito da Avalanche Estrita

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<process.h>
#include<conio.h>
#include<math.h>
int carrega_configuracao();
void Extrai_Caminho(char *,char *);
void Extrai_Nome(char *,char *);
int diff(unsigned char *origem, unsigned char *dest,int tam);
int bit_get(const unsigned char *bits, int pos);
void bit_set(unsigned char *bits, int pos, int state);
char configuracoes[5][256];
int main(void)
{
  int erro=0;
  int iNumBits,iNumBytes;
  char strModelo[4]; //capacidade para SAC de at, 32 bits
  char strSec[4]; //capacidade para SAC de at, 32 bits
  char strEnc1[4],strEnc2[4];
  char strPrincipal[4],strSecundario[4];
  char strPath[256],strNome[256];
  FILE *arq,*dif,*princ,*secun;
  long i,j,k;
  clrscr();
  erro=carrega_configuracao();
  if(erro) {
    printf("Um erro de configura‡Æo ocorreu, verifique arquivo!");
    return 1;
  }
  //abre arquivo de diferencas
  if( (dif=fopen("difer.txt","w+"))==NULL) {
    printf("Erro ao criar arquivo difer.txt!");
    return 1;
  }
  //carrega n$mero de bits e bytes
  iNumBits=atoi(configuracoes[4]);
  iNumBytes=(iNumBits-1)/8+1;
  //La‡o que gera modelo principal
  //por exemplo
  // 00 01 10 11 --> Modelos principais
  // 01 11 11    --> Modelos secund rios v lidos (maiores)
  // 10          --> Modelos secund rios v lidos (maiores)
  //    00 00 01 --> Modelos secund rios inv lidos (menores)
  //          10 --> Modelos secund rios inv lidos (menores)
  for(i=0;i< (unsigned long)((1<<iNumBits)-1);i++) {
    gotoxy(1,1);
    printf("Modelo..: %05i",i);
    //Inicializa vari vel
    //$ltimo byte , diferente
    strModelo[0]= (i << (sizeof(long)-iNumBits)) >> (sizeof(long)-8);
    for(k=1;k<iNumBytes;k++)
      strModelo[k]= ((i << (sizeof(long)-iNumBits)) >> (sizeof(long)-(k+1)*8)) &
    //Grava Modelo Principal
    if( (arq=fopen("ModPrinc.txt","wb+")) ==NULL) {
      printf("Erro ao criar ModPrinc.txt!");
      return 1;
    }
    for(k=0;k<iNumBytes;k++)
      fputc(strModelo[k],arq);
    //Verifica se varia chave ou mensagem
    if(!strcmpi("chave",configuracoes[1])) {
      //acrescentar caracteres at, completar 32
      for(k=0;k<32-iNumBytes;k++)
```

```
fputc(char(0),arq);
    }
    fclose(arq);
    //encripta modelo principal
    //Extrai_Caminho(configuracoes[0],strPath);
    //Extrai_Nome(configuracoes[0],strNome);
    if(!strcmpi("chave",configuracoes[1]))
      k=spawnl(P_WAIT,configuracoes[0],configuracoes[0],configuracoes[2],
"ModPrinc.txt","$1.txt",NULL);
    else
      k=spawnl(P_WAIT,configuracoes[0],configuracoes[0],"ModPrinc.txt",
configuracoes[3],"$1.txt",NULL);
    if(k==-1) perror(sys_errlist[errno]);
    if((princ=fopen("$1.txt","rb"))==NULL) {
  printf("Erro ao abrir arquivo $1.txt!");
  return 1;
    }
    for(k=0;k<iNumBytes;k++)
      strEnc1[k]=fgetc(princ);
    fclose(princ);
    for(j=0;j<iNumBits;j++){
      if(bit_get(strModelo,j)==0) {
memmove(strSecundario,strModelo,iNumBytes);
//altera bit
bit_set(strSecundario,j,1);
if( (arq=fopen("ModSec.txt","wb+")) ==NULL) {
  printf("Erro ao criar ModSec.txt!");
  return 1;
}
for(k=0;k<iNumBytes;k++)
  fputc(strSecundario[k],arq);
//Verifica se varia chave ou mensagem
if(!strcmpi("chave",configuracoes[1])) {
  for(k=0;k<32-iNumBytes;k++)
    fputc(char(0),arq);
}
fclose(arq);
if(!strcmpi("chave",configuracoes[1]))
  k=spawnl(P_WAIT,configuracoes[0],configuracoes[0],configuracoes[2],
"ModSec.txt","$2.txt",NULL);
else
  k=spawnl(P_WAIT,configuracoes[0],configuracoes[0],"ModSec.txt",
configuracoes[3],"$2.txt",NULL);
if(k==-1) perror(sys_errlist[errno]);
if((secun=fopen("$2.txt","rb"))==NULL) {
  printf("Erro ao abrir arquivo $2.txt!");
  return 1;
}
for(k=0;k<iNumBytes;k++)
  strEnc2[k]=fgetc(secun);
fclose(secun);
fprintf(dif,"%i\n",diff(strEnc1,strEnc2,iNumBits));
      }//if
    }//for do modelo secund rio
  }//for do modelo principal
  rewind(dif);
  //Calcula a frequencia de cada bit
  long int *freq=(long int *) malloc(sizeof(long int)*(iNumBits+1));
  for(k=0;k<=iNumBits;k++)
    freq[k]=0;
  int valor;
  long NumValores= (1<<(iNumBits-1))*iNumBits;
  for(i=0;i< NumValores;i++) {
    fscanf(dif,"%i",&valor);
    freq[valor]++;
  }
  fclose(dif);
  float media=0;
  for(i=0;i<=iNumBits;i++)
    media+=freq[i]*i;
  media/=NumValores;
  float desvio=0;
```

```
  for(i=0;i<=iNumBits;i++)
    desvio+=freq[i]*(media-i)*(media-i);
  desvio=sqrt(desvio/(NumValores-1));
  //Grava em arquivo dados estatisticos
  FILE *est;
  if( (est=fopen("estatist.csv","w") )==NULL) {
    return -1;
  }
  fprintf(est,"numbits;...freq;\n");
  //frequencias
  for(i=0;i<=iNumBits;i++)
    fprintf(est,"%7li;%7li;\n",i,freq[i]);
  fprintf(est,"Arquivo:;%s;\n",configuracoes[0]);
  fprintf(est,"Variar:;%s;\n",configuracoes[1]);
  fprintf(est,"N$mero de bits:;%i;\n",iNumBits);
  fprintf(est,"M,dia:;%6.2f;\n",media);
  fprintf(est,"Desvio:;%6.2f;\n",desvio);
  free(freq);
  fclose(est);
 return 0;
}
int carrega_configuracao(){
  FILE *conf;
  char section[256];
  //encontra arquivo
  if( (conf=fopen("config.txt","r"))==NULL)
  {
    printf("Arquivo config.txt nao encontrado!");
    return -1;
  }
  //encontra encriptador
  while(!feof(conf))
  {
    fscanf(conf,"%s",&section);
    if(!strcmpi(section,"encriptador"))
      fscanf(conf,"%s",configuracoes[0]);
    if(!strcmpi(section,"variar"))
      fscanf(conf,"%s",configuracoes[1]);
    if(!strcmpi(section,"arqplano"))
      fscanf(conf,"%s",configuracoes[2]);
    if(!strcmpi(section,"arqchave"))
      fscanf(conf,"%s",configuracoes[3]);
    if(!strcmpi(section,"numbits"))
      fscanf(conf,"%s",configuracoes[4]);
  }
  return 0;
}
void bit_set(unsigned char *bits, int pos, int state) {
unsigned char mask;
int i;
mask = 0x80;
for (i = 0; i<(pos % 8); i++ )
mask = mask >> 1;
if(state)
bits[pos/8] = bits[pos/8] | mask;
else
bits[pos/8] = bits[pos/8] & (~mask);
return;
}
int bit_get(const unsigned char *bits, int pos) {
unsigned char mask;
int i;
mask = 0x80;
for (i=0;i<(pos%8);i++)
mask = mask >>1;
return (((mask & bits[(int)(pos/8)]) == mask ) ? 1:0);
}
int diff(unsigned char *origem, unsigned char *dest,int tam)
{
```

```
  int i;
  int cont=0;
  for(i=0;i<tam;i++)
    if(bit_get(origem,i)!=bit_get(dest,i)) cont++;
  return cont;
}
void Extrai_Caminho(char *strOr, char *strDest)
{
  char *aux=strOr;
  int cont=0,temp=0,i;
  //encontra no. de '/'
  while( *aux!='\0' ) if(*aux++=='/') cont++;
  aux=strOr;
  //caminha at, barra
  while(temp<cont) {
    if(*aux=='/') temp++;
    strDest[i++]=*aux++;
  }
  strDest[i]='\0';
}
void Extrai_Nome(char *strOr, char *strDest)
{
  char *aux=strOr;
  int cont=0,temp=0,i;
  //encontra no. de '/'
  while( *aux!='\0' ) if(*aux++=='/') cont++;
  aux=strOr;
  //caminha at, barra
  while(temp<cont) {
    if(*aux++=='/') temp++;
  }
  while(*aux!='\0') strDest[i++]=*aux++;
  strDest[i]='\0';
}
```

## B.8 Criptossistema de Roskin atende ao SAC?

### Modelo de 2 bits para Criptossistema de Roskin

A figura B.1 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 2 bits. Neste caso existem 4 ocorrências que foram trocadas um bit.

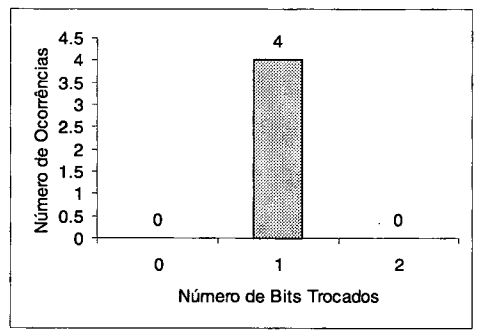


Figura B.1: Modelo de 2 bits - Variandoo texto plano

A figura B.2 ilustra o efeito sobre o texto cifrado, variando a chave de 2 bits.

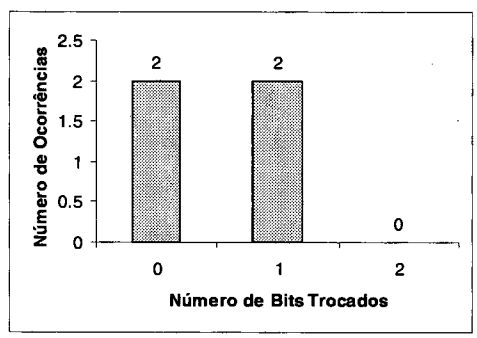


Figura B.2: Modelo de 2 bits - Variando a chave

A tabela B.2 mostra a variação da média, desvio padrão e moda calculada para o modelo de 2 bits. A moda indica a quantidade de bit que teve maior ocorrência de troca. Variando o texto aberto, observa-se que troca de um bit teve 4 ocorrências, enquanto variando a chave, nenhum ou um bit de troca teve 2 ocorrências.

Tabela B.2: Dados Analíticos para Modelo de 2 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1 | 0 | 1 | 4 |
| chave | 0,50 | 0,58 | 0 e 1 | 2 |

**Modelo de 3 bits para Criptossistema de Roskin**

Como já foi visto na seção 5.5.1, as possibilidades para um arquivo de 3 bits são 12. A figura B.3 ilustra o efeito sobre o texto cifrado, variando o texto aberto para de 3 bits.

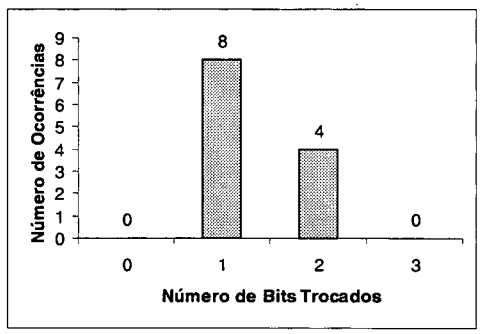


Figura B.3: Modelo de 3 bits - Variando o texto aberto

A figura B.4 ilustra o efeito sobre o texto cifrado, variando a chave de 3 bits. Variando um bit da chave, existem 4 ocorrências que não houve nenhuma troca. Em 4 casos foram trocados um bit e 4 casos, houve a troca de 2 bits.

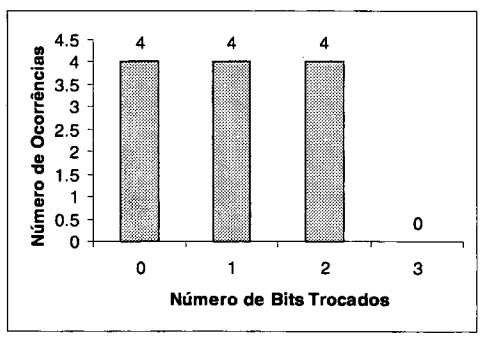


Figura B.4: Modelo de 3 bits - Variando a chave

A tabela B.3 mostra a variação da média, desvio padrão e moda calculada para o modelo de 3 bits.

Tabela B.3: Dados Analíticos para Modelo de 3 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,33 | 0,49 | 1 | 8 |
| chave | 1,00 | 0,85 | 0,1,2 | 4 |

**Modelo de 4 bits para Criptossistema de Roskin**

Para arquivos de 4 bits, existem **32** possibilidades, onde as seqüências produzidas terão um bit de diferença entre si. A figura B.5 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 4 bits. Neste modelo existem 16 ocorrências de troca de um bit e 8 para troca de 2 bits e 8 para troca de 3 bits.

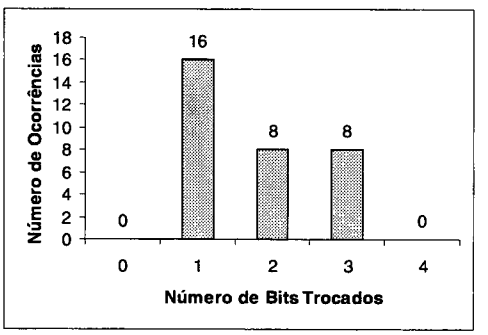


Figura B.5: Modelo de 4 bits - Variando o texto aberto

A figura B.6, ilustra o efeito sobre o texto cifrado, variando a chave de 4 bits. Neste modelo existem 24 ocorrências de troca de 2 bits e 8 ocorrências que nenhum bit foi trocado.

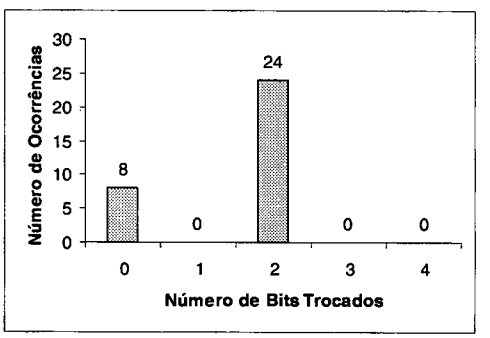


Figura B.6: Modelo de 4 bits - Variando a chave

A tabela B.4 mostra a variação da média, desvio padrão e moda calculada para o modelo de 4 bits. Os valores de média e desvio padrão são próximos.

Tabela B.4: Dados Analíticos para Modelo de 4 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,75 | 0,84 | 1 | 16 |
| chave | 1,50 | 0,88 | 2 | 24 |

**Modelo de 5 bits para Criptossistema de Roskin**

Para um arquivo de 5 bits, as possibilidades de troca de um bit são **80**. A figura B.7, ilustra o efeito sobre o texto cifrado, variando o texto aberto de 5 bits. Neste modelo existem 48 ocorrências de troca de um bit e 16 ocorrências para troca de 2 bits e 16 para troca de 3 bits.

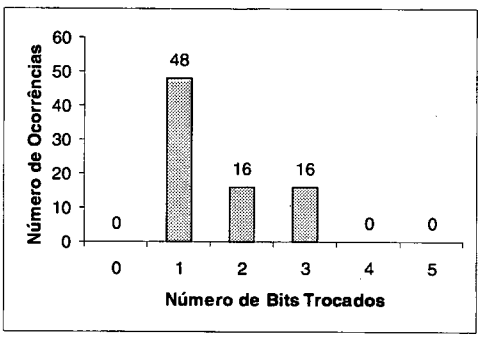


Figura B.7: Modelo de 5 bits - Variando o texto aberto

A figura B.8 mostra o efeito sobre o texto cifrado, variando a chave de 5 bits. Neste modelo existem 48 ocorrências de troca de 2 bits e 16 ocorrências que nenhum bit foi trocado e 16 ocorrências de troca de 3 bits.

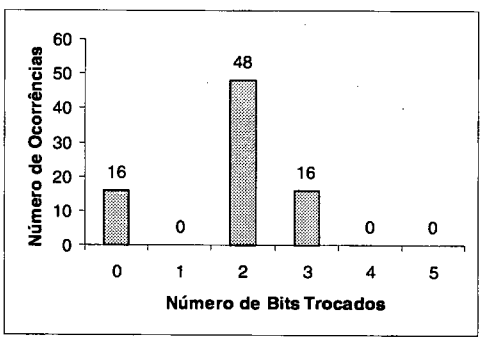


Figura B.8: Modelo de 5 bits - Variando a chave

A tabela B.5 mostra a variação da média, desvio padrão e moda calculada para o modelo de 5 bits.

Tabela B.5: Dados Analíticos para Modelo de 5 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,60 | 0,81 | 1 | 48 |
| chave | 1,80 | 0,99 | 2 | 48 |

**Modelo de 6 bits para Criptossistema de Roskin**

Para um arquivo de 6 bits, as possibilidades de troca de um bit são **192**. A figura B.9 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 6 bits. Neste modelo existem 128 ocorrências de troca de um bit e 32 ocorrências para troca de 2 bits e 32 para troca de 3 bits.

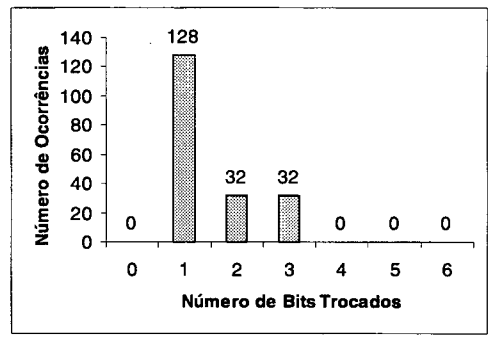


Figura B.9: Modelo de 6 bits - Variando o texto aberto

A figura B.10 ilustra o efeito sobre o texto cifrado, variando a chave de 6 bits. Neste modelo existem 48 ocorrências de troca de 2 bits e 32 ocorrências para troca de 1, 3, 4 e 5 bits. Existem também 16 ocorrência com nenhuma troca de bits.

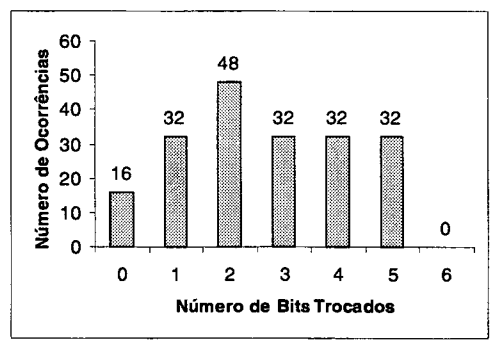


Figura B.10: Modelo de 6 bits - Variando a chave

A tabela B.6 mostra a variação da média, desvio padrão e moda calculada para o modelo de 6 bits. Neste modelo existem valores bastante diferentes no desvio padrão e na média.

Tabela B.6: Dados Analíticos para Modelo de 6 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,50 | 0,77 | 1 | 128 |
| chave | 2,67 | 1,55 | 2 | 48 |

**Modelo de 7 bits para Criptossistema de Roskin**

Para um arquivo de 7 bits, as possibilidades de troca de um bit são **448**. A figura B.11 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 7 bits. Existem 320 ocorrências de troca de um bit e 64 ocorrências para troca de 2 bits e 64 para troca de 3 bits.

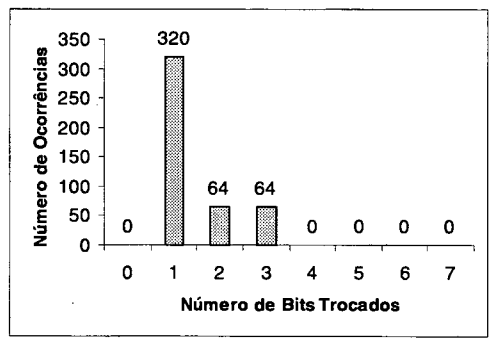


Figura B.11: Modelo de 7 bits - Variando o texto aberto

A figura B.12 ilustra o efeito sobre o texto cifrado, variando a chave de 7 bits. Neste modelo existem 112 ocorrências de troca de 3 e 4 bits e 32 ocorrências para troca de 1, 2 e 6 bits. Existem também 16 ocorrência com nenhuma troca de bits e 16 ocorrências para troca de 7 bits.

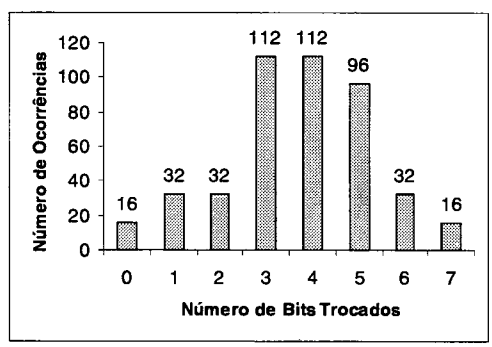


Figura B.12: Modelo de 7 bits - Variando a chave

A tabela B.7 mostra a variação da média, desvio padrão e moda calculada para o modelo de 7 bits.

Tabela B.7: Dados Analíticos para Modelo de 7 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,43 | 0,73 | 1 | 320 |
| chave | 3,71 | 1,58 | 3 e 4 | 112 |

**Modelo de 8 bits para Criptossistema de Roskin**

Para um arquivo de 8 bits, as possibilidades de troca de um bit são **1024**. A figura B.13 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 8 bits. Neste modelo existem 528 ocorrências de troca de um bit, 240 ocorrências para troca de 2 bits, 192 ocorrências para troca de 3 bits, 48 ocorrências para troca de 4 bits e 16 ocorrências para troca de 5 bits.

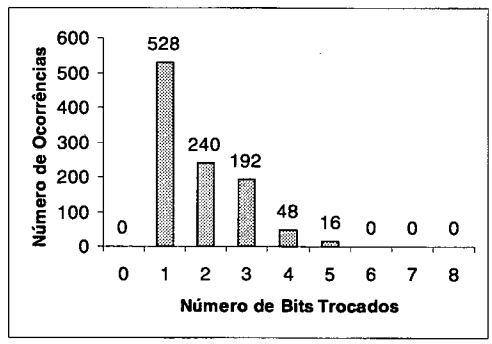


Figura B.13: Modelo de 8 bits - Variando o texto aberto

A figura B.14 ilustra o efeito sobre o texto cifrado, variando a chave de 8 bits. Neste modelo existem 320 ocorrências de troca de 4 bits e a distribuição de troca de bits começa se aproximar de uma curva gaussiana.

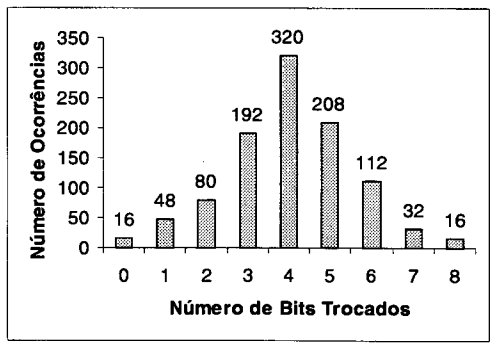


Figura B.14: Modelo de 8 bits - Variando a chave

A tabela B.8 mostra a variação da média, desvio padrão e moda calculada para o modelo de 8 bits. Neste modelo existem valores bastante diferentes no desvio padrão e na média. Variando o texto aberto, as trocas normalmente ocorrem nos primeiros bits, enquanto variando a chave, as trocas se aproximam de uma curva gaussiana.

**Modelo de 9 bits para Criptossistema de Roskin**

Para um arquivo de 9 bits, as possibilidades de troca de um bit são **2304**. A figura B.15 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 9 bits. Neste modelo existem 800 ocorrências de troca de 2 bits, 736 ocorrências para troca de 1 bit,

Tabela B.8: Dados Analíticos para Modelo de 8 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,81 | 1,00 | 1 | 528 |
| chave | 4,03 | 1,53 | 4 | 320 |

384 ocorrências para troca de 3 bits, 288 ocorrências para troca de 4 bit e 96 ocorrências para troca de 5 bits.

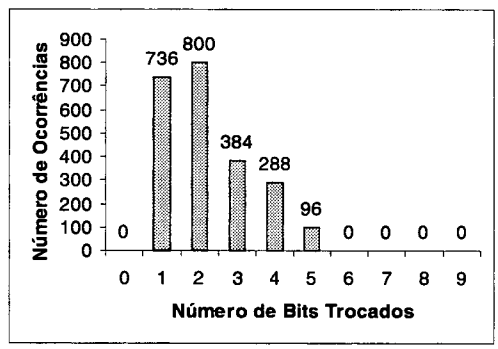


Figura B.15: Modelo de 9 bits - Variando o texto aberto

A figura B.16 ilustra o efeito sobre o texto cifrado, variando a chave de 9 bit. Neste modelo existem 608 ocorrências de troca de 4 bits e a distribuição de troca de bits começa se aproximar de uma curva gaussiana.

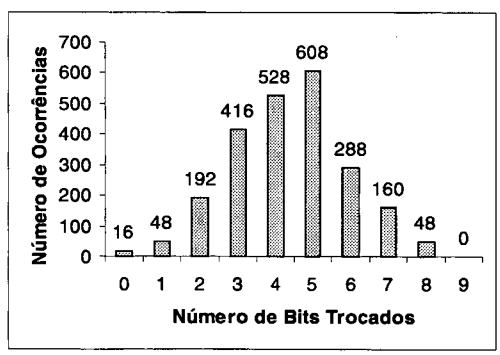


Figura B.16: Modelo de 9 bits - Variando a chave

A tabela B.9 mostra a variação da média, desvio padrão e moda calculada para o modelo de 9 bits. Neste modelo variando o texto aberto, as trocas normalmente ocorrem nos primeiros bits, enquanto variando a chave, as trocas se aproximam a uma curva gaussiana. Não há muita diferença para os valores do desvio padrão, mas existe grande diferença para valores do média.

Tabela B.9: Dados Analíticos para Modelo de 9 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 2,22 | 1,15 | 2 | 800 |
| chave | 4,37 | 1,55 | 5 | 608 |

**Modelo de 10 bits para Criptossistema de Roskin**

Para um arquivo de 10 bits, as possibilidades de troca de um bit são **5120**. A figura B.17 ilustra o efeito sobre o texto cifrado, texto aberto de 10 bits. Neste modelo variando o texto aberto, existem 1664 ocorrências de troca de 2 bits e 1280 ocorrências para troca de 1 bit. Existem 960 ocorrências para troca de 3 bits e 704 ocorrências para troca de 4 bits. As maiores ocorrências de trocas estão concentradas nos primeiros bits.

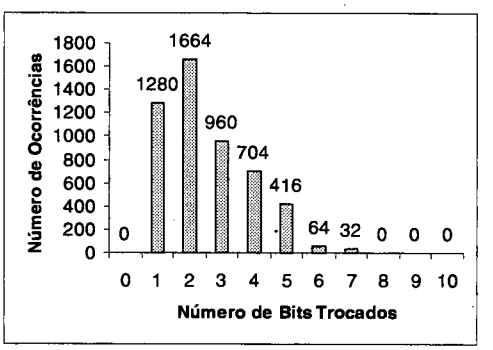


Figura B.17: Modelo de 10 bits - Variando o texto aberto

A figura B.18 ilustra o efeito sobre o texto cifrado, variando a chave de 10 bits. Neste modelo variando a chave, existem 1248 ocorrências de troca de 5 bits e a distribuição de troca de bits aproxima-se a uma curva gaussiana.

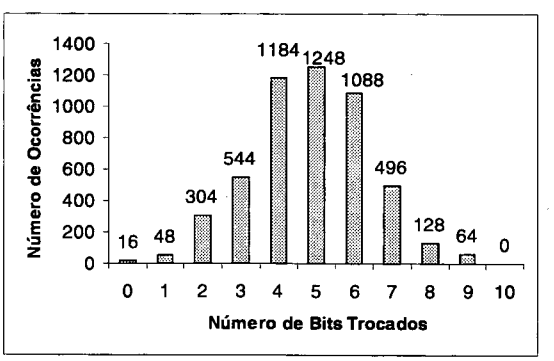


Figura B.18: Modelo de 10 bits - Variando a chave

A tabela B.10 mostra a variação da média, desvio padrão e moda calculada para o modelo de 10 bits. Neste modelo variando o texto aberto, as trocas normalmente ocorrem nos primeiros bits, enquanto variando a chave, as trocas se aproximam a

de uma curva gaussiana. Não há muita diferença para os valores do desvio padrão, mas existe grande diferença nos valores da média.

Tabela B.10: Dados Analíticos para Modelo de 10 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 2,54 | 1,34 | 2 | 1664 |
| chave | 4,86 | 1,56 | 5 | 1248 |

**Modelo de 11 bits para Criptossistema de Roskin**

Para um arquivo de 11 bits, as possibilidades de troca de um bit são **11264**. A figura B.19 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 11 bits. Neste modelo existem 2720 ocorrências de troca de 3 bits e 2496 ocorrências para troca de 1 e 2 bits. Existem 2016 ocorrências para troca de 4 bits e 1056 ocorrências para troca de 5 bits. As maiores ocorrências de trocas estão concentradas nos primeiros bits.

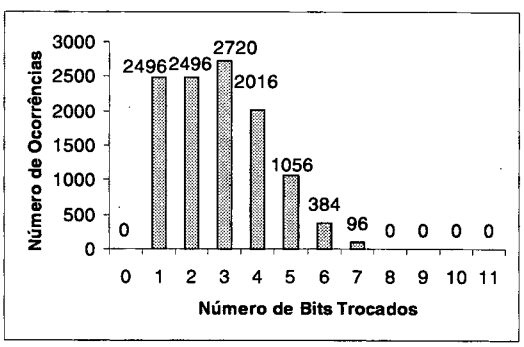


Figura B.19: Modelo de 11 bits - Variando o texto aberto

A figura B.20 ilustra o efeito sobre o texto cifrado, variando a chave de 11 bits. Neste modelo variando a chave, existem 2480 ocorrências de troca de 6 bits e a distribuição de troca de bits cada vez mais, aproxima-se a uma curva gaussiana.

A tabela B.11 mostra a variação da média, desvio padrão e moda calculada para o modelo de 11 bits. Neste modelo variando o texto aberto, as maiores ocorrências de troca ocorrem nos primeiros bits, enquanto variando a chave, as trocas se aproximam a uma curva gaussiana. Não há muita diferença para os valores do desvio padrão, mas existe grande diferença nos valores da média.

Tabela B.11: Dados Analíticos para Modelo de 11 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 2,84 | 1,44 | 3 | 2720 |
| chave | 5,29 | 1,67 | 6 | 2480 |

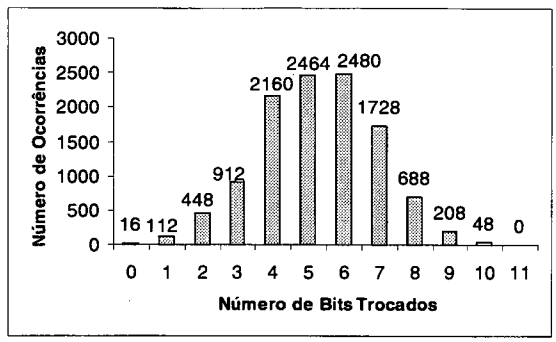


Figura B.20: Modelo de 11 bits - Variando a chave

**Modelo de 12 bits para Criptossistema de Roskin**

Para um arquivo de 12 bits, as possibilidades de troca de um bit são **24576**. A figura B.21 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 12 bits. Neste modelo variando o texto aberto, existem 5568 ocorrências de troca de 3 bits e 4992 ocorrências para troca de 2 e 4 bits. Existem 4416 ocorrências para troca de 1 bit e 2816 ocorrências para troca de 5 bits. As maiores ocorrências de troca estão concentradas nos primeiros bits.

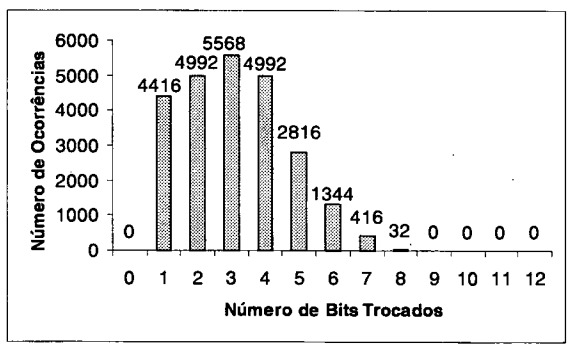


Figura B.21: Modelo de 12 bits - Variando o texto aberto

A figura B.22 ilustra o efeito sobre o texto cifrado, variando a chave de 12 bits. Neste modelo variando a chave, existem 5568 ocorrências de troca de 6 bits e a distribuição de troca de bits cada vez mais, aproxima-se a uma curva gaussiana.

A tabela B.12 mostra a variação da média, desvio padrão e moda calculada para o modelo de 12 bits. Neste modelo variando o texto aberto, as maiores ocorrências de troca ocorrem nos primeiros bits, enquanto variando a chave, as trocas se aproximam a uma curva gaussiana. Não há muita diferença para os valores do desvio padrão, mas existe grande diferença nos valores da média. A maior ocorrência de troca é igual a 5568 para variação do texto aberto ou para variação da chave.

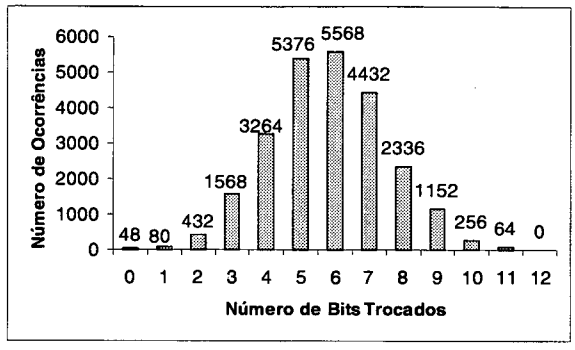


Figura B.22: Modelo de 12 bits - Variando a chave

Tabela B.12: Dados Analíticos para Modelo de 12 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 3,11 | 1,54 | 3 | 5568 |
| chave | 5,79 | 1,71 | 6 | 5568 |

**Modelo de 13 bits para Criptossistema de Roskin**

Para um arquivo de 13 bits, as possibilidades de troca de um bit são **53248**. A figura B.23 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 13 bits. Neste modelo existem 11872 ocorrências de troca de 1 bits e 11584 ocorrências para troca de 3 bits. As maiores ocorrências de troca estão concentradas nos primeiros bits.

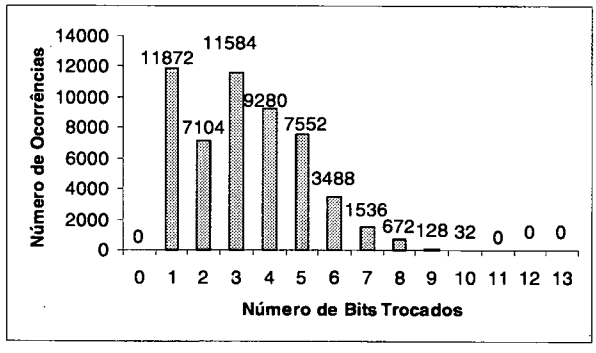


Figura B.23: Modelo de 13 bits - Variando o texto aberto

A figura B.24 ilustra o efeito sobre o texto cifrado, variando a chave de 13 bits. Neste modelo existem 11264 ocorrências de troca de 6 bits e a distribuição de troca de bits cada vez mais, aproxima-se a uma curva gaussiana.

A tabela B.13 mostra a variação da média, desvio padrão e moda calculada para o modelo de 13 bits. Neste modelo as maiores ocorrências de troca ocorrem nos primeiros bits, enquanto variando a chave, as trocas se aproximam a uma curva gaussiana. Não há muita diferença para os valores do desvio padrão, mas existe grande diferença nos valores da média.

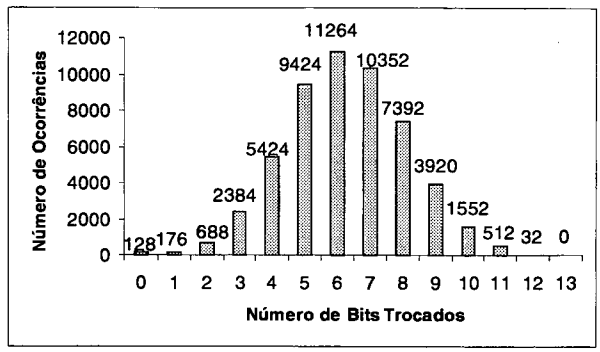


Figura B.24: Modelo de 13 bits - Variando a chave

Tabela B.13: Dados Analíticos para Modelo de 13 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 3,27 | 1,78 | 1 | 11872 |
| chave | 6,26 | 1,85 | 6 | 11264 |

**Modelo de 14 bits para Criptossistema de Roskin**

Para um arquivo de 14 bits, as possibilidades de troca de um bit são **114688**. A figura B.25 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 14 bits. Neste modelo existem 28096 ocorrências de troca de 1 bits e 19264 ocorrências para troca de 3 bits. As maiores ocorrências de troca estão concentradas nos primeiros bits.

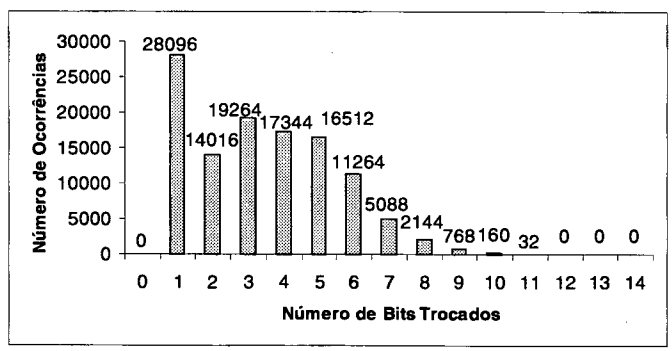


Figura B.25: Modelo de 14 bits - Variando o texto aberto

A figura B.26 ilustra o efeito sobre o texto cifrado, variando a chave de 14 bits. Neste modelo existem 23184 ocorrências de troca de 7 bits e a distribuição de troca de bits, cada vez mais, aproxima-se a uma curva gaussiana.

A tabela B.14 mostra a variação da média, desvio padrão e moda calculada para o modelo de 14 bits. Neste modelo variando o texto aberto, as maiores ocorrências de troca ocorrem nos primeiros bits, enquanto variando a chave, as trocas formam uma curva gaussiana. Não há muita diferença para os valores do desvio padrão, mas existe grande diferença nos valores da média.

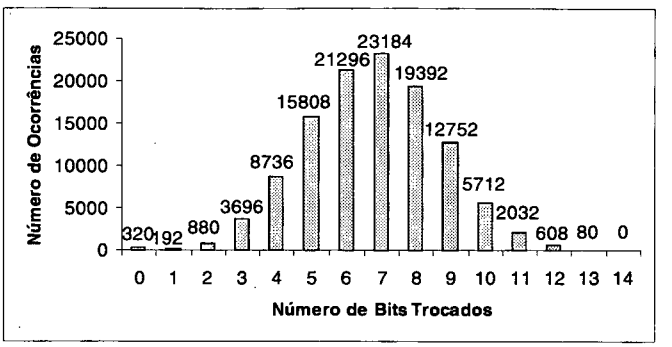


Figura B.26: Modelo de 14 bits - Variando a chave

Tabela B.14: Dados Analíticos para Modelo de 14 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 3,44 | 2,00 | 1 | 28096 |
| chave | 6,76 | 1,94 | 7 | 23184 |

**Modelo de 15 bits para Criptossistema de Roskin**

Para um arquivo de 15 bits, as possibilidades de troca de um bit são **245760**. A figura B.27 ilustra o efeito sobre o texto cifrado, variando o texto aberto de 15 bits. Neste modelo existem 69088 ocorrências de troca de 1 bits e 33408 ocorrências para troca de 5 bits.

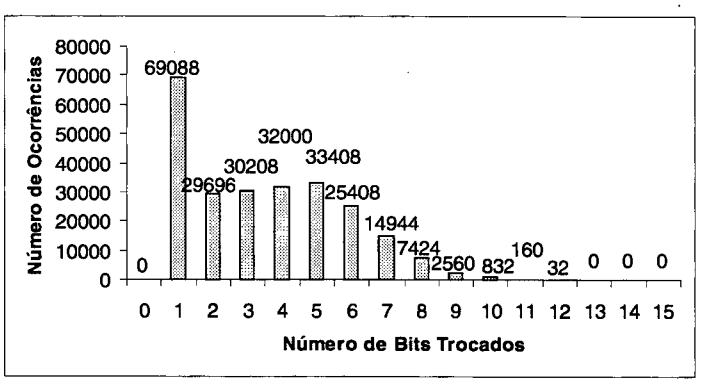


Figura B.27: Modelo de 15 bits - Variando o texto aberto

A figura B.28 ilustra o efeito sobre o texto cifrado, variando a chave de 15 bits. Neste modelo existem 47648 ocorrências de troca de 7 bits e a distribuição de troca de bits, forma uma curva gaussiana.

A tabela B.15 mostra a variação da média, desvio padrão e moda calculada para o modelo de 15 bits. Neste modelo variando o texto aberto, as maiores ocorrências de troca ocorrem nos primeiros bits, enquanto variando a chave, as trocas formam uma curva gaussiana. Não há muita diferença para os valores do desvio padrão, mas existe grande diferença nos valores da média.

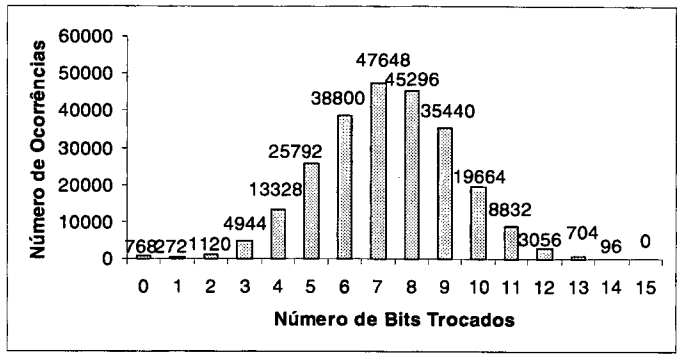


Figura B.28: Modelo de 15 bits - Variando a chave

Tabela B.15: Dados Analíticos para Modelo de 15 bits do Criptossistema Roskin

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 3,52 | 2,21 | 1 | 69088 |
| chave | 7,28 | 2,03 | 7 | 47648 |

# Apêndice C

# Código fonte do programa AES

Este apêndice contém o código fonte do programa Rijndael em assembler.

```
.radix 16
 ; RIJNDAEL cipher program (DOS) using
 ;  OFB block chaining, supporting 128,
 ;  192, and 256-bit blocks and keys and
 ;  time, date, file size, and filename
 ;  as IV. To permit proper decryption,
 ;  the file time and date are reset
 ;  to former values after writing.
 ; This program incorporates a Davies-
 ;  Meyer type key hash.
 ; The input key length is 2*blockbytes,
 ;  which more closely matches entropy
 ;  of user input to that of cipher.
 ; This version can be assembled on the
 ;  A86 shareware assembler, available
 ;  from www.eji.com and run on any '86.
 ; Copyright April 12, 2000 by
 ;  Robert G. Durnal (afn21533@afn.org)
 ;  Released for free use by anyone!
 ; Posted under license exception (TSU)
 ;  to EAR regulations.
 ; Block size in bits = 32 x NCOLS
 ; Key size in bits = 32 x NKEYS
NCOLS equ 8 ; May be 4, 6, or 8
NKEYS equ 8 ; May be 4, 6, or 8
KEYBYTES equ NKEYS*4
BLOCKBYTES equ NCOLS*4
READSIZE equ 0C000
PWR_TAB equ 0E00
LOG_TAB equ 0F00
SBOX equ LOG_TAB
STATE equ PWR_TAB
#if NCOLS GT NKEYS
ROUNDS equ 6+NCOLS
#else
ROUNDS equ 6+NKEYS
#endif
 ; Routines for RIJNDAEL S-box creation
 ;  derived from formulas in Dr. Brian
 ;  Gladman's RIJNDAEL implementation
 ; First prepare Pwr and Log tables.
 ;
 ;  for (i=0; p=1; i<256; ++i)
 ;  {
 ;     pwr_tab[i]=p; log_tab[p]=i;
```

```
;     p=p^(p<<1)^(p&0x80?0x01b:0)
;   }
MAKETABLES:
 mov di,pwr_tab
 mov bp,di      ; Save pointer to STATE
 mov bh,log_tab/100
 mov cx,100
 inc ax         ; DOS starts AX at 0000
MAKETABLOOP:
 mov bl,al
 xchg ax,di
 mov [bx],al    ; Store Log value
 xchg ax,di
 stosb          ; And Pwr value
 call xtime     ; (p<<1)^(p&0x80?0x1b:0)
 xor al,bl      ; p^(p<<1)^(p&0x80?0x1b:0)
 loop maketabloop
 ; Now use those tables to create S-box
 ;  OK to overwrite Log table.
 ;
 ;  for (i=0; i<256; ++i)
 ;  {
 ;     p=(i?(pwr_tab[255-log_tab[i]]):0);
 ;     q=p;
 ;     q=(q>>7) | (q<<1); p^=q;
 ;     q=(q>>7) | (q<<1); p^=q;
 ;     q=(q>>7) | (q<<1); p^=q;
 ;     q=(q>>7) | (q<<1); p^=q^0x63;
 ;     sbx_tab[i]=p;
 ;  }
MAKESBOX:       ; Starts with CX = 0
 mov bx,bp      ; --> PWR_TAB for XLAT
 db 0D6         ; Clears AL to start
SBOXLOOP:       ; AL = p, AH = q
 mov ah,al      ; q = p
 mov ch,4       ; CX always 04xx here
MAKES1:         ; C formula for ROL q,1
 rol ah,1       ; q = (q>>7) | (q<<1)
 xor al,ah      ; p ^= q
 dec ch
 jnz makes1     ; 4 times
 xor al,63      ; Here CH always 00
 stosb          ; Store computed value
 mov al,[di]    ; Get next log_tab[i]
 not al         ; 255-log_tab[i]
 xlat           ; pwr_tab[255-log_tab[i]]
 loop sboxloop
GETKEY:
 mov dx,askforkey
 mov ah,9
 int 21         ; Solicit via STDOUT
 mov dl,keybuf mod 100
 mov ah,0A
 int 21         ; Get key via STDIN
 mov ax,3
 int 10         ; Clear key from screen
 mov si,dx    ; SI --> KEYBUF
 lodsb
 lodsb          ; SI --> KEYARRAY
 xchg ax,di     ; DI = Length of key
 add di,si      ; DI --> CR in KEYARRAY
 mov cl,2*keybytes
PUSHKEY:        ; Push all key bytes
 push [si]      ; Only LSB will be used
 movsb          ;   and expand short keys
 loop pushkey
HASHKEY:        ; Hash the input key
 call loopinit ; Initialize DI and CL
POPKEY:         ; Pop key mat'l to CFBBUF
 pop ax
 stosb
 push bx        ; BL always 00 here
 pop bx         ; Overwrite key in stack
```

```
#if (NCOLS XOR NKEYS) AND 2 EQ 2
 cmp sp,0FFFC  ; Restrict POPs ó PUSHs
 loopnz popkey
#else
 loop popkey
 mov dx,di
 mov al,0D
 repnz scasb
 mov [di-1],ch
 rep stosb      ; Write 0Ds to DTA
OPENFILE:
 mov ax,3D02
 int 21         ; Open the file for R/W
 jc reread      ; Error opening file
 xchg ax,bx     ; Handle
GETIV:
 mov ah,4E
 int 21         ; Find First file
 call loopinit
 mov si,96      ; Time, Date, Size, Fname
 rep movsb      ;  to STATE padded w 0Ds
 mov dx,1000
REREAD: ; Loop here until read = 0 bytes
 mov si,filerrmsg
 jc exit
 mov ch,readsize/100
 mov ah,3F
 int 21
 jc exit
GOODREAD:       ; Check on read length
 or ax,ax       ; 0 length read = done
 jz endprog     ; Any RET within ñ 80h
 push dx        ; Databuf address
 mov si,dx
 push ax        ; Bytes read
 push bx        ; Handle
BLOCK:          ; Process block of data
 push ax        ; # bytes left in DATABUF
 push si        ; Databuf pointer
 call cipher    ; Encrypt STATE
 pop si         ; Databuf pointer
DATACRYPT:      ; XOR STATE & DATA
 mov al,[di]    ; From STATE
 xor [si],al    ; [SI] = newdata
 cmpsb
 loop datacrypt
 pop ax
 sub ax,blockbytes
 ja block
PTRSET: ; Decrement file pointer
 pop bx         ; Handle
 pop dx         ; Bytes read
 push dx
 neg dx
 dec cx         ; CX:DX=-(bytes read)
 mov ax,4201
 int 21 ; Back file ptr by bytes read
WRITEDATA:      ; Write # bytes read
 pop cx         ; Bytes read
 pop dx         ; Databuf address
 mov ah,40
 int 21         ; Overwrite original file
 mov si,96
 lodsw
 xchg ax,cx     ; CX is time
 lodsw
 xchg ax,dx     ; DX is date
 mov ax,5701
 int 21         ; Reset orig date/time
 jmp reread
 ; Start of subroutines
EXIT: ; Print error messages via INT29
 lodsb
```

```
 int 29
 or al,al
 jnz exit
 ret            ; INT20 exit via PSP 00
XTIME: ; Subroutine for multiplication
 ; in GF(2^8) modulo x^8+x^4+x^3+x+1
 shl al,1
 jnc endprog
 xor al,1B
ENDPROG:        ; INT 20 via PSP 00
 ret
KEYSCHED:       ; Run key schedule on key
 ; RIJNDAEL key schedule expands first
 ;  KEYBYTES of key into sufficient key
 ;  material for ROUNDS+1 key mixing
 ; Rounds+2 avoids roundoff error
 mov di,keyarray+keybytes ; Fall thru RJ
 mov dl,1       ; Initialize RCON
 mov cl,(rounds+2)*ncols/nkeys
KEYSCHEDLOOP:
 push cx        ; On entry CH = 0, saved
 mov bx,-4      ; Effective rotation of key
 cmp ch,3       ;  material achieved by
 jnz $+4        ;  index manipulation
 rcl bl,1
 adc bl,0       ;  BX = -3, -4, or -7
 mov al,[di+bx]
 test ch,-(ncols/8*10+4)
 jnz keycont    ; TEST with 0EC or 0FC
KEYSUB:
 mov bx,sbox    ; Bytesub
 xlat
 or ch,ch
 jnz keycont
 xor al,dl      ; CH = 0, mix with RCON
 xchg ax,dx
 call xtime     ; Update RCON
 xchg ax,dx
KEYCONT:        ; Final step for all keys
 xor al,[di-keybytes]
 stosb
 inc ch
 cmp ch,keybytes
 jb keyschedloop+1
 pop cx
 loop keyschedloop
CIPHER: ; The RIJNDAEL encryption cipher
 ; The 'STATE' of the cipher is treated
 ;  as a row-major byte matrix of 4 rows
 ;  by NCOLS columns for SHIFTROW and
 ;  MIXCOLUMN routines. BP is pointer
 ;  to STATE throughout, SI is pointer
 ;  to next byte of key material.
 ; This routine automatically follows
 ;  the KEYSCHED routine, or is called
 ;  directly from the BLOCK sequence.
 mov si,keyarray
 mov cx,rounds ; CH non-zero after read
 push cx
RIJNLOOP:       ; Encrypt STATE
 call loopinit ; Initialize DI and CL
 mov bx,sbox
ADDKEYLOOP:
 lodsb          ; Next key byte
 xor al,[di]
 xlat           ; Do the BYTESUB here
 stosb
 loop addkeyloop
SHIFTROW:
 mov cl,3       ; CX is row # base 0
SR_OUTER:
 mov bx,cx
#if ncols EQ 8
 cmp bx,2
```

```
 jc $+3
 inc bx         ; 1,2,3 --> 1,3,4
#endif
 dec di         ; --> Rows 3, 2, 1
SR_INNER:
 mov ch,ncols-1
 push di
 mov al,[di]
SHIFTER:        ; Shift row 1 space <--
 sub di,4       ; Move DI <-- 1 col
 xchg al,[di]
 dec ch
 jnz shifter
 pop di
 mov [di],al
 dec bx
 jnz sr_inner
 loop sr_outer
ENDSR:
 pop ax         ; Round counter
 dec ax
 jz lastaddkey ; Skip last MIXCOLUMN
 push ax        ; Restack round counter
MIXCOLUMN: ; Matrix mult in GF(2^8)
 mov di,bp      ; STATE
 mov cl,ncols
NEXTCOL:        ; Multiplier matrix is
 mov ch,4       ;  ° 2 3 1 1 °
 mov dx,[di]    ;  ° 1 2 3 1 °
 mov bx,[di+2] ;  ° 1 1 2 3 °
COLUMNLOOP:     ;  ° 3 1 1 2 °
 mov ax,dx
 call xtime     ; 2x byte 0
 xchg ah,al
 call xtime     ; 2x byte 1
 xor al,dh      ; 3x byte 1
 xor ax,bx      ; Add in bytes 2 and 3
 xor al,ah      ; AL=matrix product byte
 stosb          ; Store in original byte
 xchg dh,dl     ; Proceed to next byte
 xchg bl,dh     ;  3 XCHGs equivalent to
 xchg bh,bl     ;  ROL 8 on 32-bit reg
 dec ch
 jnz columnloop
 loop nextcol
 jmp rijnloop
LASTADDKEY:
 call loopinit ; Initialize DI and CL
LASTKEYLOOP:
 lodsb
 xor [di],al
 scasb
 loop lastkeyloop
LOOPINIT:       ; Init BP & CL for loop
 mov di,bp
 mov cl,blockbytes
 ret
ASKFORKEY: db 'Key:',24
FILERRMSG: db 'File error',0
KEYBUF: db 2*keybytes+1
KEYARRAY equ $+1 ; Leave space for count
FILESIZE equ $-100
```

# Apêndice D

# Modelos Propostos

Este apêndice contém os três modelos propostos e a biblioteca que foi utilizada para criação dos modelos propostos.

## D.1 Esquema único de cifrar e decifrar

```
#include <iostream.h>
#include <fstream.h>

void load_session_keys(ifstream& key_file,
unsigned char session_keys[]);
double init_pad(unsigned char session_keys[]);
unsigned char init_code(unsigned char session_keys[]);
double logistic(double x, int iterations);
int next_subkey(int key);
double map_to(unsigned char x);
unsigned char map_from(double x);
double chop(double x);

const double r = 3.9;

int main(int argc, char *argv[]) {
if(argc != 3) {
cout << "Uso: rosu key_file texto_plano > texto_cifrado"
                    << endl;
return 10;
}

ifstream key_file(argv[1]);
if(!key_file.is_open()) {
cerr << "Erro: Não pode abrir arquivo chave." << endl;
return 10;
}
unsigned char session_keys[32];
load_session_keys(key_file, session_keys);
key_file.close();

ifstream plaintext(argv[2]);
if(!plaintext.is_open()) {
cerr << "Erro: Não pode abrir o texto plano." << endl;
return 10;
}

double pad = init_pad(session_keys);
unsigned char code = init_code(session_keys);

unsigned char data; // the data byte
```

```
int subkey = 0; // the current subkey
int c;

while( (c = plaintext.get()) != EOF) {
data = c;
pad = logistic( chop(map_to(session_keys[subkey]) + pad ),
16 + session_keys[next_subkey(subkey)]
);

code = (int(data) ^ map_from(pad)) ;
cout << char(code);

subkey = next_subkey(subkey);
}

plaintext.close();

return 0;
}

void load_session_keys(ifstream& key_file,
unsigned char session_keys[])
{
for(int i = 0; i < 32; i++)
key_file >> session_keys[i];
}

double init_pad(unsigned char session_keys[]) {
unsigned char pad = session_keys[0];
for(int i = 1; i < 32; i++)
pad = pad ^ session_keys[i];

return double(pad) / 256.0;
}

unsigned char init_code(unsigned char session_keys[]) {
unsigned char code = session_keys[0];
for(int i = 1; i < 32; i++)
code = (int(code) + int(session_keys[i])) % 256;

return code;
}

double logistic(double x, int iterations) {
for(int i = 0; i < iterations; i++)
x = r*x*(1 - x);

return x;
}

int next_subkey(int key) {
return (key + 1 ) % 32;
}

double map_to(unsigned char x) {
return double(x) / 256.0;
}

unsigned char map_from(double x) {
return (unsigned char)(x * 256.0);
}

double chop(double x) {
return x - int(x);
}
```

## D.2 Biblioteca cripto.h

```
#define w 8
#define rotl(x,y) (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))
#define rotr(x,y) (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))
void ROTL(char *word, int iNumBits,int iBitsRot);
void ROTR(char *word, int iNumBits,int iBitsRot);
int iNumBits;
unsigned char *Gera_Modelo(unsigned char *descricao);
void Altera_Modelo(char *strModelo, char *strPadrao, int iPos);
void load_session_keys(FILE *key_file, unsigned char session_keys[]);
double init_pad(unsigned char session_keys[]);
unsigned char init_code(unsigned char session_keys[]);
double logistic(double x, int iterations);
int next_subkey(int key);
double map_to(unsigned char x);
unsigned char map_from(double x);
double chop(double x);
void criptografar(FILE *origem, FILE *dest,
unsigned char session_keys[]);
void criptoreverso(FILE *origem, FILE *dest,
unsigned char session_keys[]);
void descriptografar(FILE *origem, FILE *dest,
unsigned char session_keys[]);
void descriptoreverso(FILE *origem, FILE *dest,
unsigned char session_keys[]);
void criptonovo(FILE *, FILE *,unsigned char *);
void descriptonovo(FILE *, FILE *,unsigned char *);
int diff(unsigned char *origem, unsigned char *dest,int tam);
long int tamarq(FILE *arq){
  long pos_cur=ftell(arq);
  long pos=0;
  fseek(arq,2,0);
  pos=ftell(arq);
  fseek(arq,0,pos_cur);
  return pos;
}
const double r = 3.9; // logistic constant
int bit_get(const unsigned char *bits, int pos);
void bit_set(unsigned char *bits, int pos, int state);
void bit_set(unsigned char *bits, int pos, int state) {
unsigned char mask;
int i;
mask = 0x80;
for (i = 0; i<(pos % 8); i++ )
mask = mask >> 1;
if(state)
bits[pos/8] = bits[pos/8] | mask;
else
bits[pos/8] = bits[pos/8] & (~mask);
return;
}
int bit_get(const unsigned char *bits, int pos) {
unsigned char mask;
int i;
mask = 0x80;
for (i=0;i<(pos%8);i++)
mask = mask >>1;
return (((mask & bits[(int)(pos/8)]) == mask ) ? 1:0);
}

unsigned char *Gera_Modelo(unsigned char *descricao)
{
  int i,j,k;
  char *strAux,*strModelo,strPadrao[128];
  int iNumPadroes,iPos,iNumRep,iNumBits;
  //le numero de bits
```

```
  sscanf(descricao,"%i",&iNumBits);
  //Aloca memoria para armazenar o modelo
  strModelo = (char *) malloc(iNumBits/8+1);
  //inicializa strModelo com todos os bits iguais a 0
  strAux = strModelo;
  //avanca ate proxima posicao de leitura
  while(*(descricao++)!='\n');
  for(i=0;i<iNumBits/8;i++)
*(strAux++) = 0;
  //Laco externo para leitura de todos os modelos dos arquivos
  //Le numero de padroes
  sscanf(descricao,"%d",&iNumPadroes);
  //Inicializa apontador de posicao a ser alterada
  iPos=0;
  //para cada padrao monta o padrao de bits
  for(i=0; i<iNumPadroes;i++)
  {
//Avanca ate proxima leitura
while(*(descricao++)!=' ');
//Le a string padrao
sscanf(descricao,"%s",strPadrao);
//Avanca ate proxima leitura
while(*(descricao++)!=' ');
//Le o numero de repeticoes
sscanf(descricao,"%i",&iNumRep);
for(j=0;j<iNumRep;j++)
{
  Altera_Modelo(strModelo,strPadrao,iPos);
  iPos+=strlen(strPadrao);
} //Fim do Padrao
  }//Fim do Modelo
  return strModelo;
}

void Altera_Modelo(char *strModelo, char *strPadrao, int iPos)
{
  int i;
  for(i=0;i<strlen(strPadrao);i++)
bit_set(strModelo,iPos+i,strPadrao[i]-'0');
}

void criptoreverso(FILE *origem, FILE *dest,
unsigned char session_keys[])
{
  //inicializacao
  double pad = init_pad(session_keys);// the starting pad
  unsigned char code = init_code(session_keys);// the starting code
  unsigned char data; // the data byte
  int subkey = 0; // the current subkey
  //clrscr();
  long int i;
  fpos_t tamarq;
  fseek(origem,0,SEEK_END);
  fgetpos(origem,&tamarq);
  for(i=tamarq-1;i>=0;i--)
  {//varre origem criptografando
fseek(origem,i,SEEK_SET);
data = fgetc(origem);
pad = logistic(chop(map_to(session_keys[subkey]) + pad),
16 + session_keys[next_subkey(subkey)] + code);
code = ( (int)data + map_from(pad)) % 256;// apply the pad
putc((char)code,dest);// write out the code
subkey = next_subkey(subkey);// use next subkey
  }
}
void criptografar(FILE *origem, FILE *dest,
unsigned char session_keys[])
{
  //inicializacao
  double pad = init_pad(session_keys);// the starting pad
  unsigned char code = init_code(session_keys);// the starting code
  unsigned char data; // the data byte
  int subkey = 0; // the current subkey
```

```
  //clrscr();
  long int i=0;
  fpos_t tamarq;
  fseek(origem,0,SEEK_END);
  fgetpos(origem,&tamarq);
  fseek(origem,0,SEEK_SET);
  for(;i<tamarq;i++)
  {//varre origem criptografando
data = fgetc(origem);
pad = logistic(chop(map_to(session_keys[subkey]) + pad),
16 + session_keys[next_subkey(subkey)] + code);
code = ((int)data + map_from(pad)) % 256;// apply the pad
putc((char)code,dest);// write out the code
subkey = next_subkey(subkey);// use next subkey
  }
}
void criptonovo(FILE *origem, FILE *dest,
unsigned char session_keys[])
{
  //inicializacao
  double pad = init_pad(session_keys);// the starting pad
  unsigned char code = init_code(session_keys);// the starting code
  unsigned char data; // the data byte
  int subkey = 0; // the current subkey
  //clrscr();
  long int i=0;
  fpos_t tamarq;
  fseek(origem,0,SEEK_END);
  fgetpos(origem,&tamarq);
  fseek(origem,0,SEEK_SET);
  for(;i<tamarq;i++)
  {//varre origem criptografando
data = fgetc(origem);
pad = logistic(chop(map_to(session_keys[subkey]) + pad),
16 + session_keys[next_subkey(subkey)] + code);
code = ( (int)data ^ map_from(pad));// apply the pad
putc((char)code,dest);// write out the code
subkey = next_subkey(subkey);// use next subkey
  }
}
void descriptonovo(FILE *origem, FILE *dest,
unsigned char session_keys[])
{
  //inicializacao
  double pad = init_pad(session_keys);// the starting pad
  unsigned char code = init_code(session_keys);// the starting code
  unsigned char data=code; // the data byte
  int subkey = 0; // the current subkey
  //clrscr();
  long int i=0;
  fpos_t tamarq;
  fseek(origem,0,SEEK_END);
  fgetpos(origem,&tamarq);
  fseek(origem,0,SEEK_SET);
  for(;i<tamarq;i++)
  {//varre origem criptografando
pad = logistic(chop(map_to(session_keys[subkey]) + pad),
16 + session_keys[next_subkey(subkey)] + data);
data = fgetc(origem);
code = ( (int)data ^ map_from(pad));// apply the pad
putc((char)code,dest);// write out the code
subkey = next_subkey(subkey);// use next subkey
  }
}
void descriptografar(FILE *origem, FILE *dest,
unsigned char session_keys[])
{
  //inicializacao
  double pad = init_pad(session_keys);// the starting pad
  unsigned char data=init_code(session_keys);// the starting code
```

```
  unsigned char code;
  int subkey = 0; // the current subkey
  int c;
  //clrscr();
  long int i=0;
  fpos_t tamarq;
  fseek(origem,0,SEEK_END);
  fgetpos(origem,&tamarq);
  fseek(origem,0,SEEK_SET);
  for(;i<tamarq;i++)
  {//varre origem criptografando
pad = logistic(chop(map_to(session_keys[subkey]) + pad),
16 + session_keys[next_subkey(subkey)] + data);
data = fgetc(origem);
code = ((int)data-map_from(pad)) % 256;// apply the pad
putc((char)code,dest);// write out the code
subkey = next_subkey(subkey);// use next subkey
  }
}
void descriptoreverso(FILE *origem, FILE *dest,
unsigned char session_keys[])
{
  //inicializacao
  double pad = init_pad(session_keys);// the starting pad
  unsigned char data=init_code(session_keys);// the starting code
  unsigned char code;
  int subkey = 0; // the current subkey
  int c;
  //clrscr();
  long int i=0;
  fpos_t tamarq;
  fseek(origem,0,SEEK_END);
  fgetpos(origem,&tamarq);
  for(i=tamarq-1;i>=0;i--)
  {//varre origem criptografando
fseek(origem,i,SEEK_SET);
pad = logistic(chop(map_to(session_keys[subkey]) + pad),
16 + session_keys[next_subkey(subkey)] + data);
data = fgetc(origem);
code = ((int)data-map_from(pad)) % 256;// apply the pad
putc((char)code,dest);// write out the code
subkey = next_subkey(subkey);// use next subkey
  }
}

void load_session_keys(FILE *key_file,
unsigned char session_keys[])
{
for(int i = 0; i < 32; i++)
session_keys[i]=fgetc(key_file);
}

// caluate the starting pad from the session keys
double init_pad(unsigned char session_keys[]) {
unsigned char pad = session_keys[0];
for(int i = 1; i < 32; i++)
pad = pad ^ session_keys[i];

return double(pad) / 256.0;
}

// caluate the starting code from the session keys
unsigned char init_code(unsigned char session_keys[]) {
unsigned char code = session_keys[0];
for(int i = 1; i < 32; i++)
code = (int(code) + int(session_keys[i])) ;

return code % 256;
}

// eval the logistic map starting from x, iterations times
double logistic(double x, int iterations) {
for(int i = 0; i < iterations; i++)
x = r*x*(1 - x);
```

```
return x;
}

// returns the next subkey after the given one
int next_subkey(int key) {
return (key + 1 ) % 32;
}

// maps a byte to [0,1] interval
double map_to(unsigned char x) {
return (double)x / 256.0;
}

// maps the [0,1] interval to a byte
unsigned char map_from(double x) {
return (unsigned char)(x * 256.0);
}

// return x with the interger part
double chop(double x) {
return x - int(x);
}

int diff(unsigned char *origem, unsigned char *dest,int tam)
{
  int i;
  int cont=0;
  for(i=0;i<tam;i++)
    if(bit_get(origem,i)!=bit_get(dest,i)) cont++;
  return cont;
}

void ROTL(char *word, int iNumBits,int iBitsRot)
{
  //Rotaciona a esquerda
  int i,pas;
  int prim_bit,bit;
  //Laco controla NUMERO DE ROTACOES A FAZER
  for(pas=0;pas<iBitsRot;pas++) {
    //armazena primeiro bit
    prim_bit = bit_get(word,0);
    //Realiza 1(uma) rotacao
    for(i=0;i<iNumBits-1;i++) {
      //armazena proximo bit
      bit = bit_get(word,i+1);
      //posiciona-o a esquerda
      bit_set(word,i,bit);
    }
    //coloca o primeiro bit no final da palavra
    bit_set(word,i,prim_bit);
  }
}

void ROTR(char *word, int iNumBits,int iBitsRot)
{
  //Rotaciona a direita
  int i,pas;
  int ult_bit,bit;
  //Laco controla NUMERO DE ROTACOES A FAZER
  for(pas=0;pas<iBitsRot;pas++) {
    //armazena ultimo bit
    ult_bit = bit_get(word,iNumBits-1);
    //Realiza 1(uma) rotacao
    for(i=iNumBits-1;i>0;i--) {
      //armazena proximo bit
      bit = bit_get(word,i-1);
      //posiciona-o a esquerda
      bit_set(word,i,bit);
    }
    //coloca o ultimo bit no inicio da palavra
    bit_set(word,0,ult_bit);
  }
}

void XOR(char *str1, char *str2, int iNumBits)
```

```
{
  //Realiza um XOR bit a bit e armazena o resultado em str1
  int i,pas;
  int bit;
  for(i=0;i<iNumBits;i++) {
    bit_set(str1,i,bit_get(str1,i) ^ bit_get(str2,i) );
  }
}
void AND(char *str1, char *str2, int iNumBits)
{
  //Realiza um AND bit a bit e armazena o resultado em str1
  int i,pas;
  int ult_bit,bit;
  for(i=0;i<iNumBits;i++) {
    bit_set(str1,i,bit_get(str1,i) & bit_get(str2,i) );
  }
}
void OR(char *str1, char *str2, int iNumBits)
{
  //Realiza um OR bit a bit e armazena o resultado em str1
  int i;
  for(i=0;i<iNumBits;i++) {
    bit_set(str1,i,bit_get(str1,i) | bit_get(str2,i) );
  }
}
void criptonovo(unsigned char *origem,
unsigned char *dest,unsigned char session_keys[])
{
  //inicializacao
  double pad = init_pad(session_keys);// the starting pad
  unsigned char code = init_code(session_keys);// the starting code
  unsigned char data; // the data byte
  int subkey = 0; // the current subkey
  //clrscr();
  int i;
  for(i=0;i<=iNumBits/8;i++)
  {//varre origem criptografando
    data = origem[i];
    pad = logistic(
chop(map_to(rotr(session_keys[subkey],3)) + pad),
16 + rotr(session_keys[next_subkey(subkey)],3) + code
);
    data = data ^ session_keys[subkey];
    data = rotr(data,3);
    data = data ^ session_keys[next_subkey(subkey)];
    data = rotr(data,3);
    code = (int(data) + map_from(pad)) % 256;// apply the pad
    dest[i]=(char(code));// write out the code
    subkey = next_subkey(subkey);// use next subkey
  }
}
```

## D.3 Esquema do cifrador Reverso

```
#include<conio.h>
#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<IO.h>
#include"cripto.h"
#include<process.h>
void main(int argc, void *argv[])
{
  clrscr();
  FILE *or;
  FILE *key;
  FILE *dst;
  FILE *temp;
  char chave[32];
  if((or=fopen((const char*) argv[1],"rb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[1]);
getch();
exit(1);
  }
  if((key=fopen((const char*) argv[2],"rb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[2]);
getch();
exit(1);
  }
  if((dst=fopen((const char*) argv[3],"wb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[3]);
getch();
exit(1);
  }
  if((temp=fopen("$$$.txt","wb+"))==NULL){
printf("Arquivo $$$.txt");
getch();
exit(1);
  }
  load_session_keys(key,chave);
  criptografar(or,temp,chave);
  criptoreverso(temp,dst,chave);
  fclose(key);
  fclose(or);
  fclose(dst);
}
```

## D.4 Esquema do decifrador Reverso

```
#include<conio.h>
#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<IO.h>
#include"cripto.h"
#include<process.h>
void main(int argc, void *argv[])
{
  clrscr();
  FILE *or;
  FILE *key;
  FILE *dst;
  FILE *temp;
  char chave[32];
  if((or=fopen((const char*) argv[1],"rb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[1]);
getch();
exit(1);
  }
  if((key=fopen((const char*) argv[2],"rb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[2]);
getch();
exit(1);
  }
  if((dst=fopen((const char*) argv[3],"wb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[3]);
getch();
exit(1);
  }
  if((temp=fopen("$$$.txt","wb+"))==NULL){
printf("Arquivo $$$.txt");
getch();
exit(1);
  }
  load_session_keys(key,chave);
  descriptografar(or,temp,chave);
  descriptoreverso(temp,dst,chave);
  fclose(key);
  fclose(or);
  fclose(dst);
}
```

## D.5 Esquema do cifrador proposto

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
#include<IO.h>
#include"cripto.h"
#include<process.h>
void main(int argc, void *argv[])
{
  clrscr();
  FILE *or;
  FILE *key;
  FILE *dst;
  unsigned char chave[32];
  if((or=fopen((const char*) argv[1],"rb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[1]);
getch();
exit(1);
  }
  if((key=fopen((const char*) argv[2],"rb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[2]);
getch();
exit(1);
  }
  if((dst=fopen((const char*) argv[3],"wb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[3]);
getch();
exit(1);
  }
  load_session_keys(key,chave);
  criptonovo(or,dst,chave);
  fclose(key);
  fclose(or);
  fclose(dst);
}
```

## D.6 Esquema do decifrador proposto

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<string.h>
#include<IO.h>
#include"cripto.h"
#include<process.h>
void main(int argc, void *argv[])
{
  clrscr();
  FILE *or;
  FILE *key;
  FILE *dst;
  unsigned char chave[32];
  if((or=fopen((const char*) argv[1],"rb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[1]);
getch();
exit(1);
  }
  if((key=fopen((const char*) argv[2],"rb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[2]);
getch();
exit(1);
  }
  if((dst=fopen((const char*) argv[3],"wb+"))==NULL){
printf("Arquivo %s nao encontrado!",(const char*) argv[3]);
getch();
exit(1);
  }
  load_session_keys(key,chave);
  descriptonovo(or,dst,chave);
  fclose(key);
  fclose(or);
  fclose(dst);
}
```

# Apêndice E

# Experimentos Realizados com Modelos Propostos

Este apêndice contém os experimentos realizados com os dois modelos propostos referente ao SAC.

## E.1 Criptossistema XOR atende SAC?

Para esta finalidade, foi utilizado programa SAC para cada número específico de bits, que foi desenvolvido em C e está no apêndice B.

### Modelo de 2 bits para Criptossistema XOR

A figura E.1 mostra o efeito sobre o texto cifrado com o modelo XOR, variando o texto aberto de 2 bits. Neste modelo existem 4 ocorrências quando 1 bit é trocado.

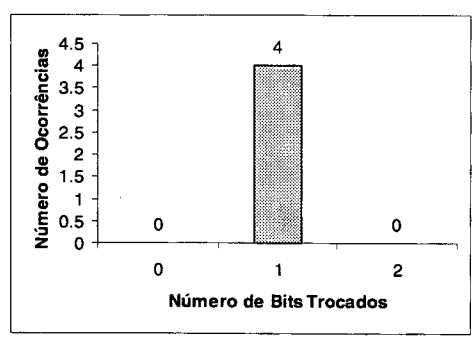


Figura E.1: XOR de 2 bits - Variando o texto aberto

A figura E.2 ilustra o efeito sobre o texto cifrado com o modelo XOR, variando a chave de 2 bits. Neste modelo existem 2 ocorrências quando nenhum bit é trocado e existem 2 ocorrências quando 2 bits são trocados.

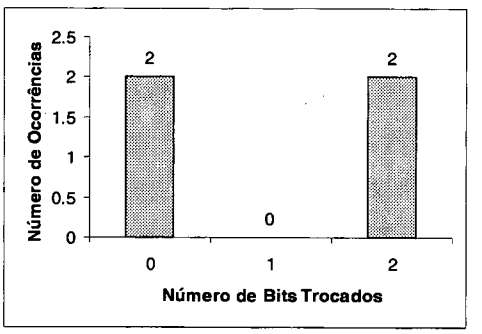


Figura E.2: XOR de 2 bits - Variando a chave

A tabela E.1 mostra a variação da média, desvio padrão e moda calculada para o modelo de 2 bits. Este modelo apresenta o mesmo valor para a média, variando o texto aberto ou a chave.

Tabela E.1: Dados Analíticos para Modelo de 2 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,00 | 0,00 | 1 | 4 |
| chave | 1,00 | 1,15 | 0 e 2 | 2 |

## Modelo de 3 bits para Criptossistema XOR

A figura E.3 ilustra o efeito sobre o texto cifrado com o modelo XOR, variando o texto aberto de 3 bits. Neste modelo existem 12 ocorrências quando 1 bit é trocado.

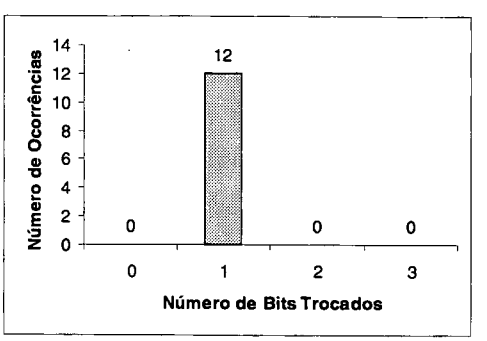


Figura E.3: XOR de 3 bits - Variando o texto aberto

A figura E.4 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave de 3 bits. Neste modelo existem 4 ocorrências quando nenhum bit é trocado, 4 ocorrências para troca de um bit, e existem 4 ocorrências quando 4 bits são trocados.

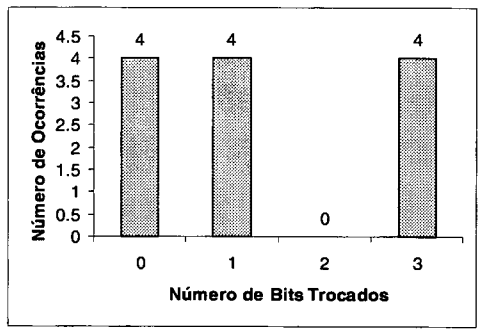


Figura E.4: XOR de 3 bits - Variando a chave

A tabela E.2 mostra a variação da média, desvio padrão e moda calculada para o modelo de 3 bits. Este modelo variando o texto aberto ou a chave, apresenta valores próximos para a média.

Tabela E.2: Dados Analíticos para Modelo de 3 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,00 | 0,00 | 1 | 12 |
| chave | 1,33 | 1,30 | 0, 1 e 3 | 4 |

**Modelo de 4 bits para Criptossistema XOR**

A figura E.5 ilustra a variação e freqüência do texto cifrado produzido pelo modelo XOR, para um texto aberto de 4 bits, variando o texto aberto. Neste modelo existem 32 ocorrências quando 1 bit é trocado.

Na figura E.6, pode-se verificar a variação do texto cifrado produzido pelo criptossistema XOR, para um arquivo de 4 bits, variando a chave. Neste modelo existem 16 ocorrências para troca de 3, 8 ocorrências para troca de 2 bits, e existem 4 ocorrências quando nenhum bit foi trocado.

A tabela E.3 mostra a variação da média, desvio padrão e moda calculada para o modelo de 4 bits. Este modelo, variando o texto aberto, apresenta as ocorrências de troca para um bit apenas. Entretanto, variando a chave, as ocorrências de troca são mais distribuídas.

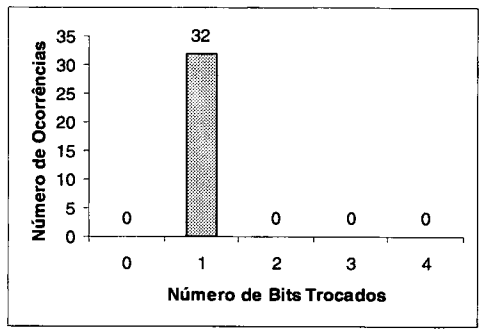


Figura E.5: XOR de 4 bits - Variando o texto aberto

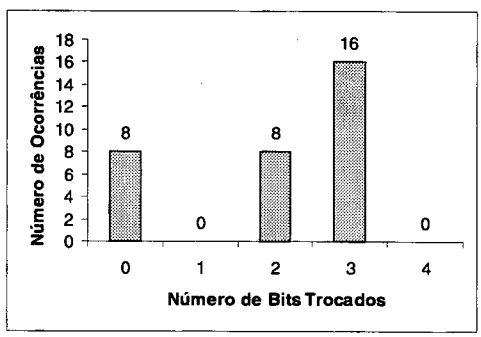


Figura E.6: XOR de 4 bits - Variando a chave

Tabela E.3: Dados Analíticos para Modelo de 4 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,00 | 0,00 | 1 | 32 |
| chave | 2,00 | 1,24 | 3 | 16 |

**Modelo de 5 bits para Criptossistema XOR**

A figura E.7 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 5 bits. Neste modelo, existem 80 ocorrências quando 1 bit é trocado. As ocorrências de troca estão concentradas no bit 1.

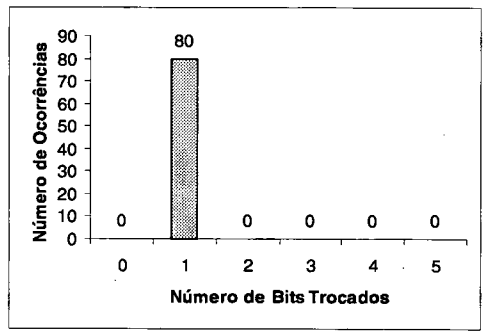


Figura E.7: XOR de 5 bits - Variando o texto aberto

A figura E.8 ilustra o efeito sobre o texto cifrado com o cifrado proposto, variando a chave de 5 bits. Neste modelo existem 32 ocorrências para troca de 2 bits, 16 ocorrências para troca de 3 e 4 bits, e existem 16 ocorrências quando nenhum bit foi trocado.

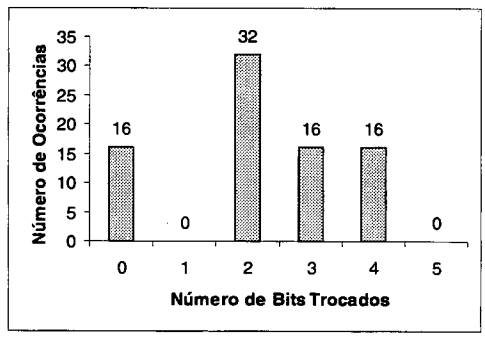


Figura E.8: XOR de 5 bits - Variando a chave

A tabela E.4 mostra a variação da média, desvio padrão e moda calculada para o modelo de 5 bits. Este modelo, variando o texto aberto, apresenta as ocorrências de troca para um bit apenas. , variando a chave, as ocorrências de troca são mais distribuídas.

Tabela E.4: Dados Analíticos para Modelo de 5 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,00 | 0,00 | 1 | 80 |
| chave | 2,20 | 1,34 | 2 | 32 |

**Modelo de 6 bits para Criptossistema XOR**

A figura E.9 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 6 bits. Neste modelo existem 192 ocorrências quando 1 bit é trocado. As ocorrências de troca estão concentradas no bit 1.

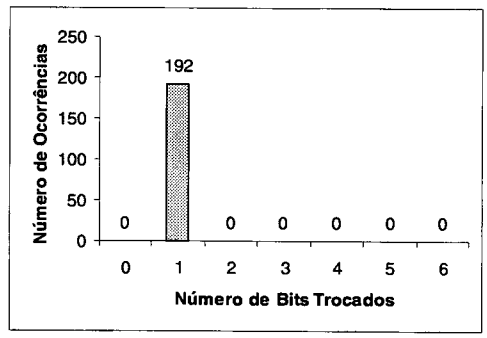


Figura E.9: XOR de 6 bits - Variando o texto aberto

A figura E.10 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave 6 bits. Neste modelo existem 48 ocorrências para troca de 2, 3 e 4 bits, 16 ocorrências para troca de 1 e 5 bits, e existem 16 ocorrências quando nenhum bit foi trocado.

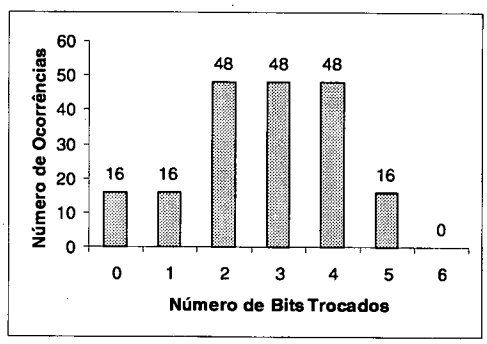


Figura E.10: XOR de 6 bits - Variando a chave

A tabela E.5 mostra a variação da média, desvio padrão e moda calculada para o modelo de 6 bits. Este modelo apresenta as ocorrências de troca para um bit apenas. No entanto, variando a chave, as ocorrências de troca são mais distribuídas.

Tabela E.5: Dados Analíticos para Modelo de 6 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,00 | 0,00 | 1 | 192 |
| chave | 2,75 | 1,37 | 2, 3 e 4 | 48 |

**Modelo de 7 bits para Criptossistema XOR**

A figura E.11 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 7 bits. Neste modelo existem 448 ocorrências quando 1 bit é trocado. As ocorrências de troca estão concentradas no bit 1.

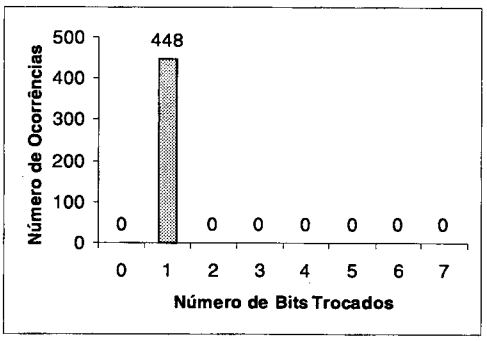


Figura E.11: XOR de 7 bits - Variando o texto aberto

A figura E.12 o efeito sobre o texto cifrado com o cifrador proposto, variando a chave de 7 bits. Neste modelo existem 128 ocorrências para troca de 4 bits, 80 ocorrências para troca de 2, 3 e 5 bits.

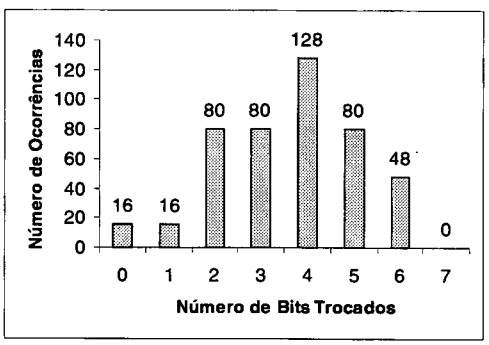


Figura E.12: XOR de 7 bits - Variando a chave

A tabela E.6 mostra a variação da média, desvio padrão e moda calculada para o modelo de 7 bits. Este modelo, variando o texto aberto, apresenta as ocorrências de troca para um bit apenas. Mas variando a chave, as ocorrências de troca são mais distribuídas.

Tabela E.6: Dados Analíticos para Modelo de 7 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,00 | 0,00 | 1 | 448 |
| chave | 3,61 | 1,50 | 4 | 128 |

**Modelo de 8 bits para Criptossistema XOR**

A figura E.13 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 8 bits. Neste modelo existem 1024 ocorrências quando 1 bit é trocado. As ocorrências de troca estão concentradas no bit 1.

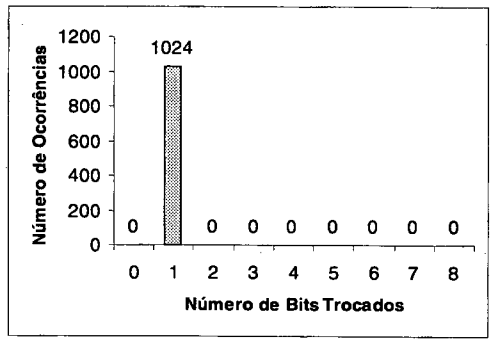


Figura E.13: XOR de 8 bits - Variando o texto aberto

A figura E.14 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave de 8 bits. Neste modelo existem 288 ocorrências para troca de 5 bits, 240 ocorrências para troca de 3 bits. As ocorrências de troca não estão distribuídos de forma uniforme.

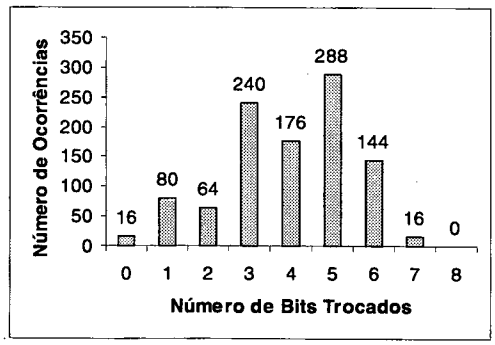


Figura E.14: XOR de 8 bits - Variando a chave

A tabela E.7 mostra a variação da média, desvio padrão e moda calculada para o modelo de 8 bits. Este modelo, variando o texto aberto, apresenta as ocorrências de troca para um bit apenas. No entretanto, variando a chave, as ocorrências de troca são mais distribuídas, mas não de forma uniforme.

**Modelo de 9 bits para Criptossistema XOR**

A figura E.15 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 9 bits. Neste modelo existem 1216 ocorrências quando

Tabela E.7: Dados Analíticos para Modelo de 8 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,00 | 0,00 | 1 | 1024 |
| chave | 3,95 | 1,56 | 5 | 288 |

1 bit é trocado, 1088 ocorrências para troca de 2 bits. As ocorrências de troca estão concentradas nos primeiros bits.

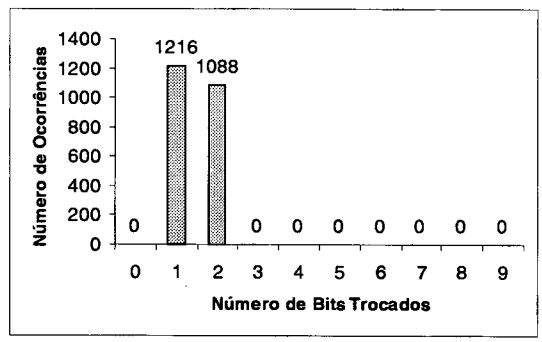


Figura E.15: XOR de 9 bits - Variando o texto aberto

A figura E.16 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave de 9 bits. Neste modelo existem 544 ocorrências para troca de 5 bits. As ocorrências de troca estão distribuídos aproximadamente de forma uniforme.

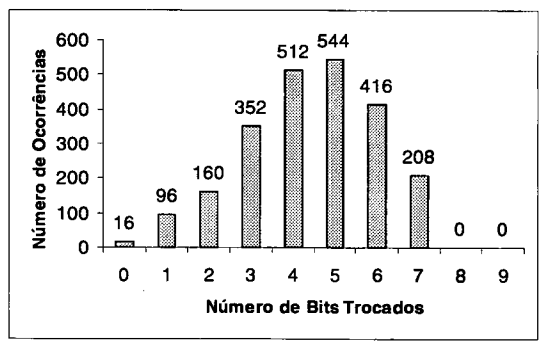


Figura E.16: XOR de 9 bits - Variando a chave

A tabela E.8 mostra a variação da média, desvio padrão e moda calculada para o modelo de 9 bits. Este modelo, variando o texto aberto, concentra as ocorrências de troca para os primeiros bits. No entanto, variando a chave, as ocorrências de troca se aproximam a uma distribuição uniforme.

Tabela E.8: Dados Analíticos para Modelo de 9 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,47 | 0,50 | 1 | 1216 |
| chave | 4,42 | 1,58 | 5 | 544 |

**Modelo de 10 bits para Criptossistema XOR**

A figura E.17 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 10 bits. Neste modelo existem 2112 ocorrências para troca de 2 bits, 1984 ocorrências para troca de 1 bit e 1024 ocorrências para troca de 3 bits. As ocorrências de troca estão concentradas nos primeiros bits.

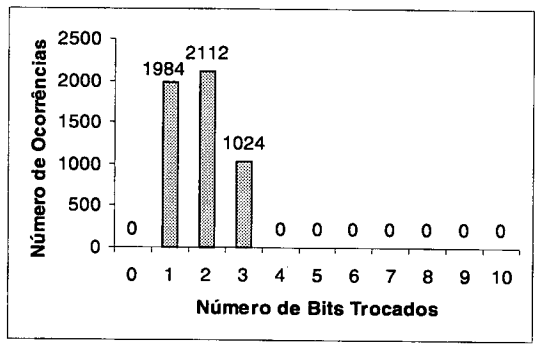


Figura E.17: XOR de 10 bits - Variando o texto aberto

A figura E.18 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave. Neste modelo existem 1104 ocorrências para troca de 5 bits. As ocorrências de troca estão distribuídos aproximadamente de forma uniforme.

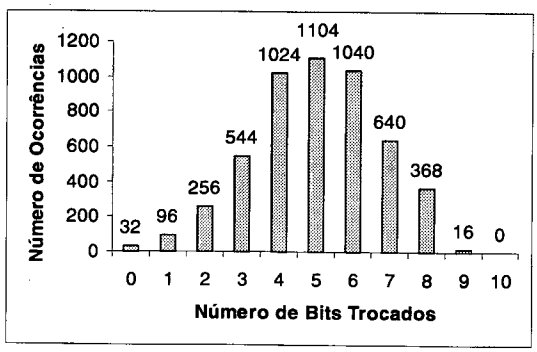


Figura E.18: XOR de 10 bits - Variando a chave

A tabela E.9 mostra a variação da média, desvio padrão e moda calculada para o modelo de 10 bits. Este modelo, variando o texto aberto, concentra as ocorrências de troca para os primeiros bits. Todavia, variando a chave, as ocorrências de troca se aproximam a uma distribuição uniforme.

Tabela E.9: Dados Analíticos para Modelo de 10 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,81 | 0,74 | 2 | 2112 |
| chave | 5,01 | 1,71 | 5 | 1104 |

**Modelo de 11 bits para Criptossistema XOR**

A figura E.19 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 11 bits. Neste modelo existem 3968 ocorrências para troca de 1 bit, 3328 ocorrências para troca de 3 bits, 2944 ocorrências para troca de 2 bits e 1024 ocorrências para troca de 4 bits. As ocorrências de troca estão concentradas nos primeiros bits.

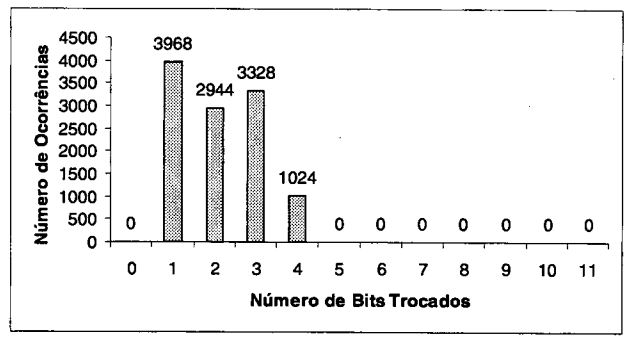


Figura E.19: XOR de 11 bits - Variando o texto aberto

A figura E.20 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave de 11 bits. Neste modelo existem 2528 ocorrências para troca de 5 bits. As ocorrências de troca estão distribuídos aproximadamente de forma uniforme.

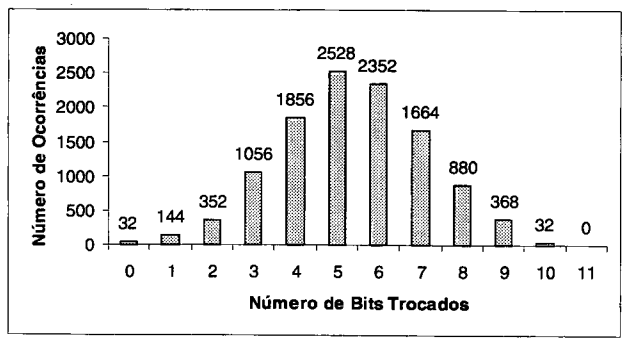


Figura E.20: XOR de 11 bits - Variando a chave

A tabela E.10 mostra a variação da média, desvio padrão e moda calculada para o modelo de 10 bits. Este modelo, variando o texto aberto, concentra as

ocorrências de troca para os primeiros bits. Todavia, variando a chave, as ocorrências de troca se aproximam a uma distribuição uniforme.

Tabela E.10: Dados Analíticos para Modelo de 11 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 2,12 | 1,00 | 1 | 3968 |
| chave | 5,37 | 1,75 | 5 | 2528 |

**Modelo de 12 bits para Criptossistema XOR**

A figura E.21 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 12 bits. Neste modelo existem 8960 ocorrências para troca de 1 bit, 6656 ocorrências para troca de 3 bits, 3584 ocorrências para troca de 4 bits, 4352 ocorrências para troca de 2 bits e 1024 ocorrências para troca de 5 bits. As ocorrências de troca estão concentradas nos primeiros bits.

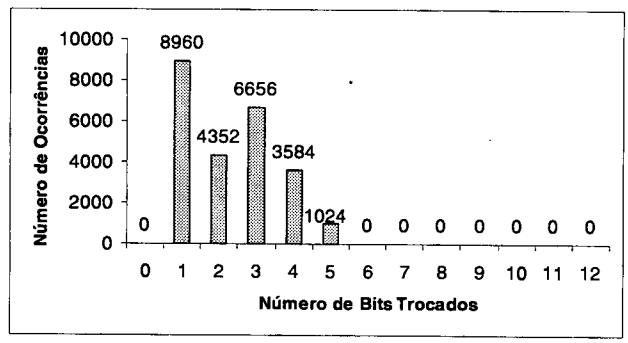


Figura E.21: XOR de 12 bits - Variando o texto aberto

A figura E.22 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave de 12 bits. Neste modelo existem 5152 ocorrências para troca de 6 bits. As ocorrências de troca estão distribuídos de forma uniforme e apresentam aproximadamente uma curva gaussiana.

A tabela E.11 mostra a variação da média, desvio padrão e moda calculada para o modelo de 12 bits. Este modelo, variando o texto aberto, concentra as ocorrências de troca para os primeiros bits. Mas variando a chave, as ocorrências de troca formam uma distribuição uniforme e aproximam-se a uma curva gaussiana.

Tabela E.11: Dados Analíticos para Modelo de 12 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 2,32 | 1,22 | 1 | 8960 |
| chave | 5,88 | 1,81 | 6 | 5152 |

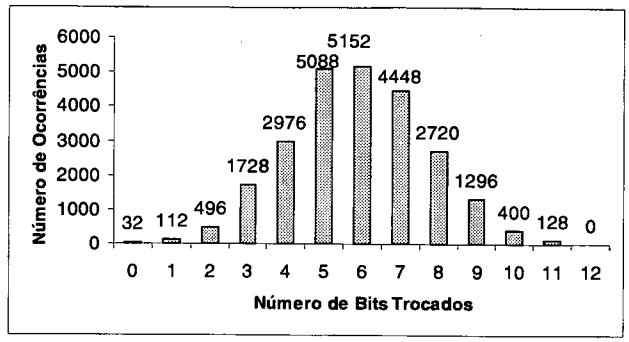


Figura E.22: XOR de 12 bits - Variando a chave

**Modelo de 13 bits para Criptossistema XOR**

A figura E.23 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 13 bits. Neste modelo existem 21504 ocorrências para troca de 1 bit, 12288 ocorrências para troca de 3 bits, 11776 ocorrências para troca de 4 bits, 3584 ocorrências para troca de 2 bits, 3072 ocorrências para troca de 5 bits e 1024 ocorrências para troca de 6 bits. As ocorrências de troca estão concentradas nos primeiros bits.

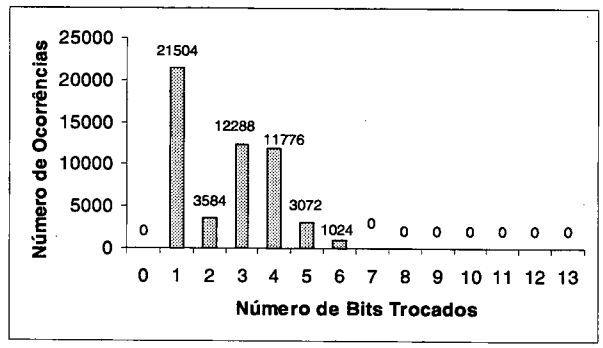


Figura E.23: XOR de 13 bits - Variando o texto aberto

A figura E.24 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave de 13 bits. Neste modelo existem 10944 ocorrências para troca de 6 bits. As ocorrências de troca estão distribuídos de forma uniforme e apresentam aproximadamente uma curva gaussiana.

A tabela E.12 mostra a variação da média, desvio padrão e moda calculada para o modelo de 13 bits. Este modelo, variando o texto aberto, concentra as ocorrências de troca para os primeiros bits. Porém, variando a chave, as ocorrências de troca formam uma distribuição uniforme e aproximam-se a uma curva gaussiana.

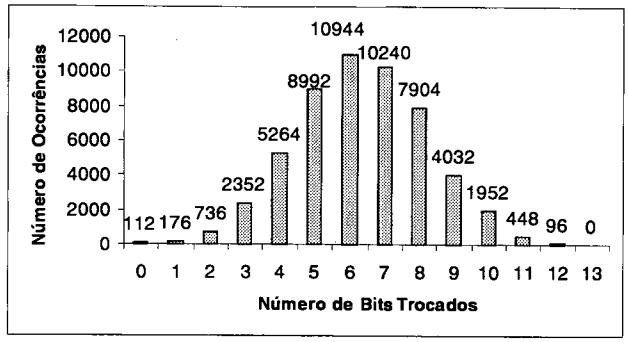


Figura E.24: XOR de 13 bits - Variando a chave

Tabela E.12: Dados Analíticos para Modelo de 13 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 2,52 | 1,44 | 1 | 21504 |
| chave | 6,33 | 1,88 | 6 | 10944 |

**Modelo de 14 bits para Criptossistema XOR**

A figura E.25 ilustra o efeito sobre o texto cifrado para o cifrador proposto, variando o texto aberto de 14 bits. Neste modelo existem 51200 ocorrências para troca de 1 bit, 23552 ocorrências para troca de 4 bits, 15360 ocorrências para troca de 3 e 5 bits, 6144 ocorrências para troca de 6 bits, 2048 ocorrências para troca de 2 bits e 1024 ocorrências para troca de 7 bits. As ocorrências de troca estão concentradas nos primeiros bits.

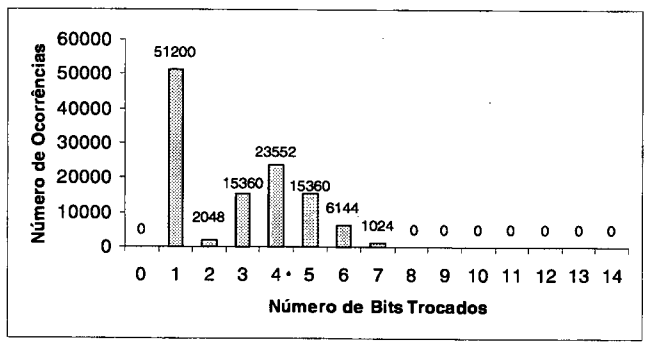


Figura E.25: XOR de 14 bits - Variando o texto aberto

A figura E.26 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave de 14 bits. Neste modelo existem 22928 ocorrências para troca de 7 bits. As ocorrências de troca estão distribuídos de forma uniforme e apresentam aproximadamente uma curva gaussiana.

A tabela E.13 mostra a variação da média, desvio padrão e moda calculada para o modelo de 14 bits. Este modelo, variando o texto aberto, concentra as

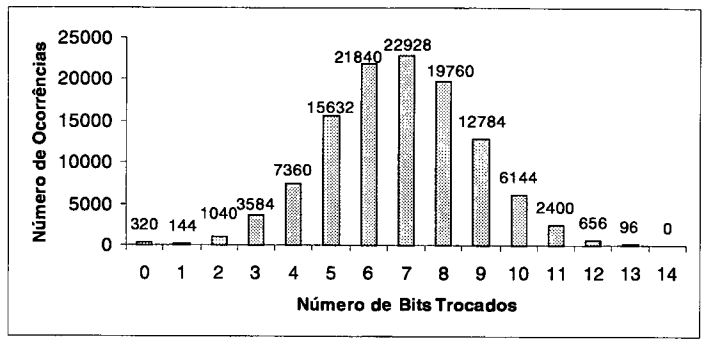


Figura E.26: XOR de 14 bits - Variando a chave

ocorrências de troca para os primeiros bits. Todavia, variando a chave, as ocorrências de troca formam uma distribuição uniforme e aproximam-se a uma curva gaussiana.

Tabela E.13: Dados Analíticos para Modelo de 14 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 2,76 | 1,76 | 1 | 51200 |
| chave | 6,82 | 1,95 | 7 | 22928 |

**Modelo de 15 bits para Criptossistema XOR**

A figura E.27 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando o texto aberto de 15 bits. Neste modelo, existem 118784 ocorrências para troca de 1 bit, 43008 ocorrências para troca de 4 bits, 34816 ocorrências para troca de 5 bits, 22528 ocorrências para troca de 6 bits, 18432 ocorrências para troca de 3 bits, 6144 ocorrências para troca de 7 bits e 2048 ocorrências para troca de 2 bits. As ocorrências de troca estão concentradas nos primeiros bits.

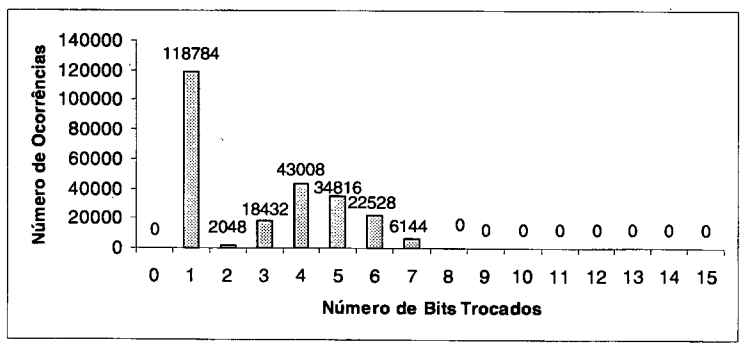


Figura E.27: XOR de 15 bits - Variando o texto aberto

A figura E.28 ilustra o efeito sobre o texto cifrado com o cifrador proposto, variando a chave de 15 bits. Neste modelo existem 46800 ocorrências para troca

de 7 bits. As ocorrências de troca estão distribuídos de forma uniforme e apresentam aproximadamente uma curva gaussiana.

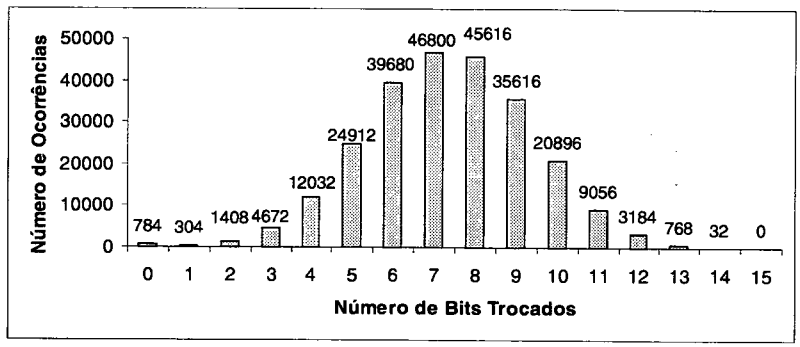


Figura E.28: XOR de 15 bits - Variando a chave

A tabela E.14 mostra a variação da média, desvio padrão e moda calculada para o modelo de 15 bits. Este modelo, variando o texto aberto, concentra as ocorrências de troca para os primeiros bits. Todavia, variando a chave, as ocorrências de troca formam uma distribuição uniforme e aproximam-se a uma curva gaussiana.

Tabela E.14: Dados Analíticos para Modelo de 15 bits do Criptossistema XOR

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 2,86 | 1,97 | 1 | 118784 |
| chave | 7,32 | 2,03 | 7 | 46800 |

## E.2 Criptossistema Reverso atende SAC?

Para poder verificar se o criptossistema reverso atende o critério SAC, foram feitos os mesmos experimentos já realizado com criptossistema de Roskin, utilizando o programa **SAC**, que encontra-se no apêndice B.

### Modelo de 2 bits para Criptossistema Reverso

A figura E.29 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 2 bits. Como a figura mostra, existem 2 ocorrências para trocas de 1 e 2 bits.

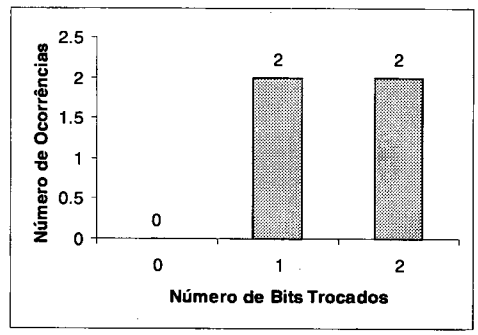


Figura E.29: Reverso de 2 bits - Variando texto o aberto

A figura E.30 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando chave de 2 bits. Como a figura mostra, existem 2 ocorrências que nenhum bit foi trocado e 2 ocorrências que 1 bit foi trocado.

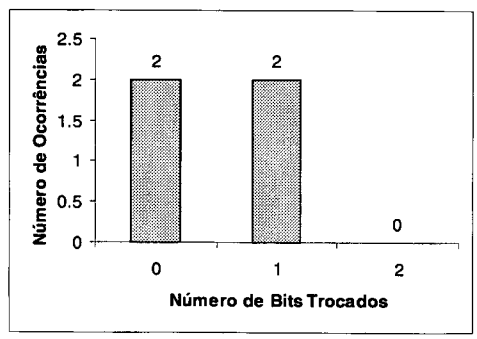


Figura E.30: Reverso de 2 bits - Variando a chave

A tabela E.15 mostra a variação da média, desvio padrão e moda calculada para o modelo reverso de 2 bits. O modelo reverso de 2 bits, variando o texto aberto ou a chave, apresenta o mesmo valor para o desvio padrão.

Tabela E.15: Dados Analíticos para Modelo de 2 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,50 | 0,58 | 1 e 2 | 2 |
| chave | 0,50 | 0,58 | 0 e 1 | 2 |

**Modelo de 3 bits para Criptossistema Reverso**

Como já foi visto na seção 5.5.1, as possibilidades para um arquivo de 3 bits são **12**. A figura E.31 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 3 bits. Neste modelo existem 4 ocorrências para trocas de 1, 2 e 3 bits.

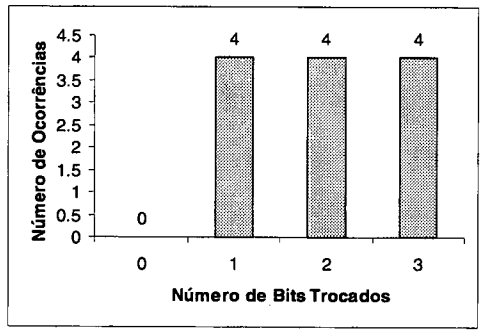


Figura E.31: Reverso de 3 bits - Variando o texto aberto

A figura E.32 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando a chave de 3 bits. Neste modelo existem 4 ocorrências que nenhum bit foi trocado, 4 ocorrências que 1 bit foi trocado e 4 ocorrências que 2 bits foram trocados.

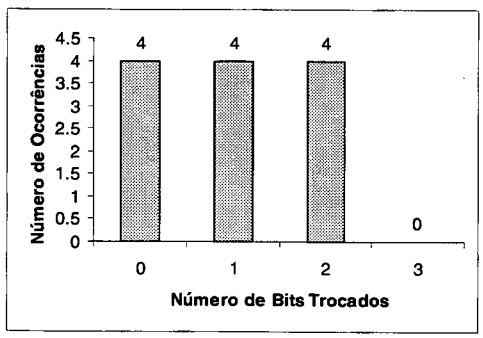


Figura E.32: Reverso de 3 bits - Variando a chave

A tabela E.16 mostra a variação da média, desvio padrão e moda calculada para o modelo de 3 bits. Este modelo variando o texto aberto ou a chave, apresenta o mesmo valor para o desvio padrão.

Tabela E.16: Dados Analíticos para Modelo de 3 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 2,00 | 0,85 | 1, 2 e 3 | 4 |
| chave | 1,00 | 0,85 | 0, 1 e 2 | 4 |

**Modelo de 4 bits para Criptossistema Reverso**

Para arquivos de 4 bits, existem **32** possibilidades, onde as seqüências produzidas terão um bit de diferença entre si. A figura E.33 mostra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 4 bits. Neste modelo existem 16 ocorrências para troca de 1 bit, 8 ocorrências que para troca de 2 e 3 bits.

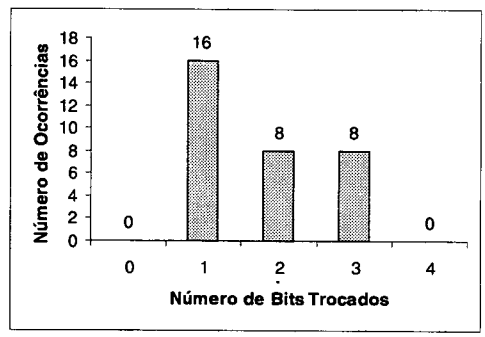


Figura E.33: Reverso de 4 bits - Variando o texto aberto

Na figura E.34, pode-se verificar o efeito sobre o texto cifrado com o cifrador reverso, a chave de 4 bits. Neste modelo existem 16 ocorrências para troca de 2 bits, 8 ocorrências para troca de 1 bit e 8 ocorrências que nenhum bit foi trocado.

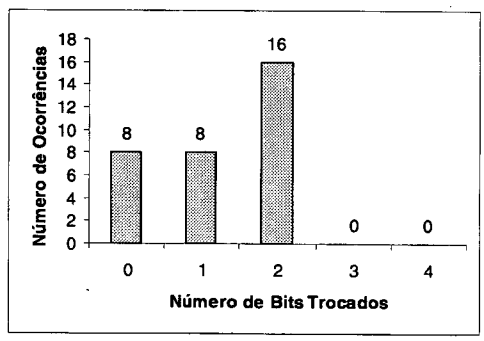


Figura E.34: Reverso de 4 bits - Variando a chave

A tabela E.17 mostra a variação da média, desvio padrão e moda calculada para o modelo de 4 bits. Este modelo apresenta valores bastantes próximo para o desvio padrão, variando o texto aberto ou a chave. Os valores obtidos para média são diferentes.

Tabela E.17: Dados Analíticos para Modelo de 4 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,75 | 0,84 | 1 | 16 |
| chave | 1,25 | 0,85 | 2 | 16 |

**Modelo de 5 bits para Criptossistema Reverso**

Para um arquivo de 5 bits, as possibilidades de troca de um bit são **80**. A figura E.35 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 5 bits. Neste modelo existem 48 ocorrências para troca de 1 bit e 16 ocorrências que para troca de 2 e 3 bits.

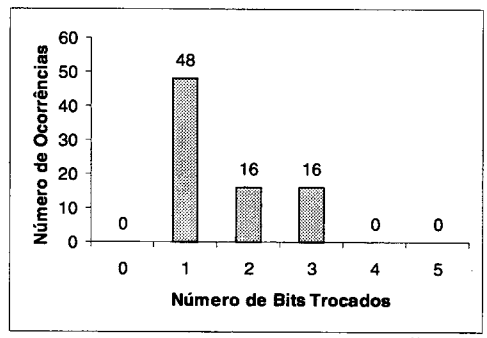


Figura E.35: Reverso de 5 bits - Variando o texto aberto

A figura E.36 ilustra o efeito sobre o texto cifrado com o cifrador reverso, a chave de 5 bits. Neste modelo existem 48 ocorrências para troca de 2 bits e 16 ocorrências para troca de 1 bit e 16 ocorrências que nenhum bit foi trocado.

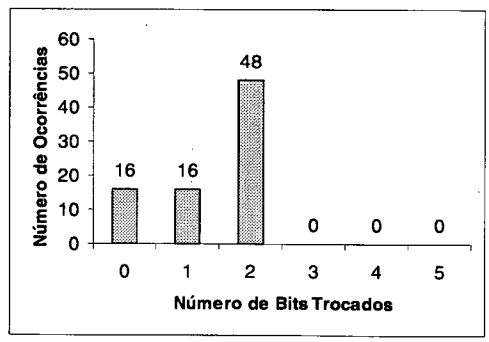


Figura E.36: Reverso de 5 bits - Variando a chave

A tabela E.18 mostra a variação da média, desvio padrão e moda calculada para o modelo de 5 bits. Este modelo apresenta o mesmo valor para o desvio padrão, variando o texto aberto ou a chave. Os valores obtidos para média são próximos.

Tabela E.18: Dados Analíticos para Modelo de 5 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,60 | 0,81 | 1 | 48 |
| chave | 1,40 | 0,81 | 2 | 48 |

**Modelo de 6 bits para Criptossistema Reverso**

Para um arquivo de 6 bits, as possibilidades de troca de um bit são **192**. A figura E.37 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 6 bits.

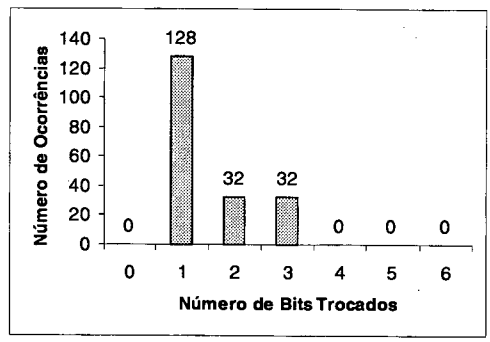


Figura E.37: Reverso de 6 bits - Variando o texto aberto

A figura E.38 ilustra o efeito sobre o texto cifrado com o cifrador reverso, a chave de 6 bits. Neste modelo existem 96 ocorrências para troca de 2 bits, 32 ocorrências para troca de 3 e 4 bits, 16 ocorrências que nenhum bit foi trocado e 16 ocorrências para troca de 5 bits.

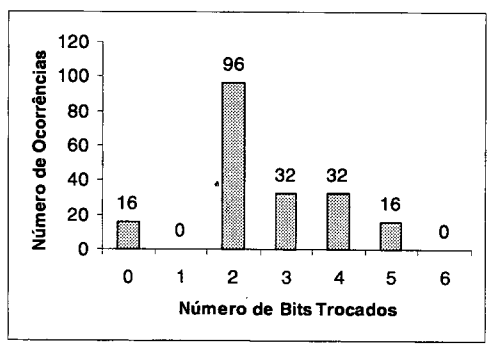


Figura E.38: Reverso de 6 bits - Variando a chave

A tabela E.19 mostra a variação da média, desvio padrão e moda calculada para o modelo de 6 bits. Este modelo apresenta valores diferentes para o desvio padrão e a média, variando o texto aberto ou a chave.

Tabela E.19: Dados Analíticos para Modelo de 6 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,50 | 0,77 | 1 | 128 |
| chave | 2,58 | 1,26 | 2 | 96 |

**Modelo de 7 bits para Criptossistema Reverso**

Para um arquivo de 7 bits, as possibilidades de troca de um bit são **448**. A figura E.39 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 7 bits. Neste modelo existem 208 ocorrências para troca de 1 bit, 128 ocorrências que para troca de 2 bits, 80 ocorrências para troca de 3 bits e 16 ocorrências para troca de 4 e 5 bits.

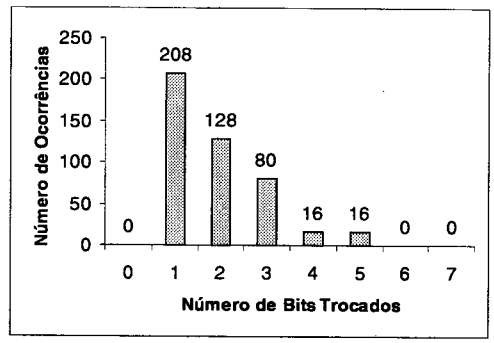


Figura E.39: Reverso de 7 bits - Variando o texto aberto

A figura E.39 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando a chave de 7 bits. Neste modelo existem existem 112 ocorrências para troca de 3 e 4 bits. 96 ocorrências para troca de 2 bits, 48 ocorrências para troca de 6 bits. As maiores ocorrências de troca estão concentradas no meio.

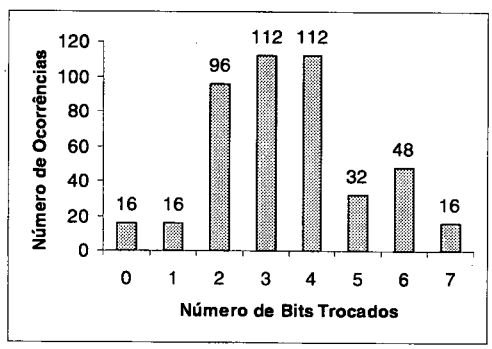


Figura E.40: Reverso de 7 bits - Variando a chave

A tabela E.20 mostra a variação da média, desvio padrão e moda calculada para o modelo de 7 bits. Este modelo, variando o texto aberto, apresenta a maior

ocorrência de troca nos primeiros bits, enquanto variando a chave, as maiores ocorrências de troca estão concentradas no meio.

Tabela E.20: Dados Analíticos para Modelo de 7 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,89 | 1,05 | 1 | 208 |
| chave | 3,46 | 1,59 | 3 e 4 | 112 |

**Modelo de 8 bits para Criptossistema Reverso**

Para um arquivo de 8 bits, as possibilidades de troca de um bit são **1024**. A figura E.41 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 8 bits. Neste modelo existem 544 ocorrências para troca de 1 bit, 256 ocorrências que para troca de 2 bits, 160 ocorrências para troca de 3 bits e 32 ocorrências para troca de 4 e 5 bits.

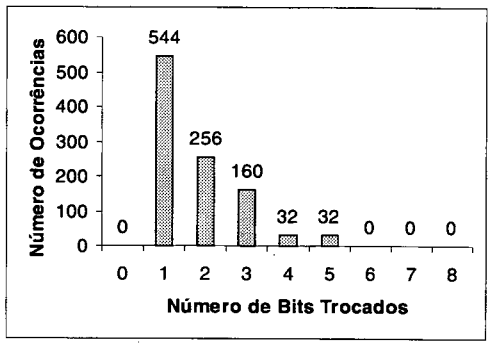


Figura E.41: Reverso de 8 bits - Variando o texto aberto

A figura E.42 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando a chave de 8 bits. Neste modelo existem 320 ocorrências para troca de 4 bits. A distribuição de ocorrências começa a se aproximar a uma curva gaussiana.

A tabela E.21 mostra a variação da média, desvio padrão e moda calculada para o modelo de 8 bits. Este modelo apresenta a maior ocorrência de troca nos primeiros bits, enquanto variando a chave, a distribuição de ocorrências de troca começa se aproximar a uma curva gaussiana.

Tabela E.21: Dados Analíticos para Modelo de 8 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 1,78 | 1,02 | 1 | 544 |
| chave | 3,89 | 1,39 | 4 | 320 |

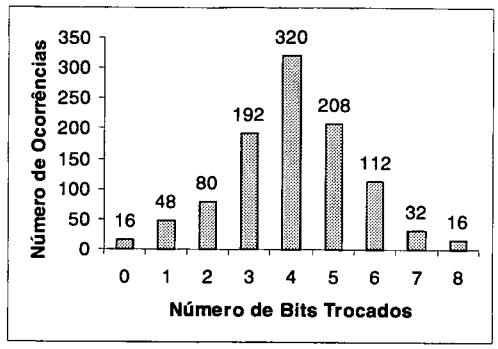


Figura E.42: Reverso de 8 bits - Variando a chave

**Modelo de 9 bits para Criptossistema Reverso**

Para um arquivo de 9 bits, as possibilidades de troca de um bit são **2304**. A figura E.43 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 9 bits. Neste modelo existem 496 ocorrências para troca de 4 bits. As ocorrências de troca de bit aumentam até chegar a 4 bits. A partir de 5 bits, começam a diminuir. Para troca de 9 bits, não há nenhuma ocorrência.

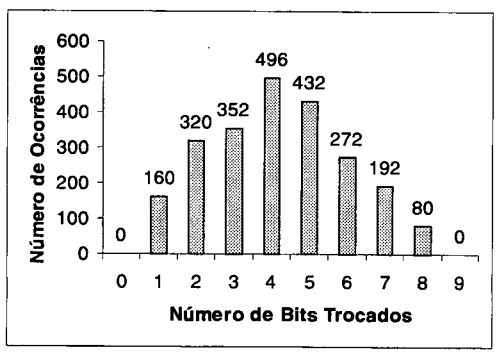


Figura E.43: Reverso de 9 bits - Variando o texto aberto

A figura E.44 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando a chave de 9 bits. Neste modelo existem 736 ocorrências para troca de 5 bits. As ocorrências de bits aumentam até chegar a 5 bits. Estas ocorrências diminuem bastante a partir de 6 bits. Não há nenhuma ocorrência para troca de 9 bits.

A tabela E.22 mostra a variação da média, desvio padrão e moda calculada para o modelo de 9 bits. Este modelo, variando o texto aberto ou a chave, apresenta uma maior distribuição de ocorrências de troca, a partir do meio, para os bits menores e maiores. Esta distribuição é mais uniforme, variando o texto aberto, pois as ocorrências de troca são mais próximos do que as ocorrências de troca, quando a chave é variada.

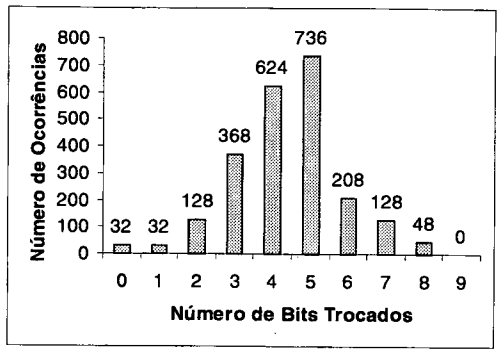


Figura E.44: Reverso de 9 bits - Variando a chave

Tabela E.22: Dados Analíticos para Modelo de 9 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 4,17 | 1,81 | 4 | 496 |
| chave | 4,38 | 1,45 | 5 | 736 |

**Modelo de 10 bits para Criptossistema Reverso**

Para um arquivo de 10 bits, as possibilidades de troca de um bit são **5120**. A figura E.45 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 10 bits. Neste modelo existem 1008 ocorrências para troca de 6 bits. A distribuição de ocorrências de troca não é uniforme.

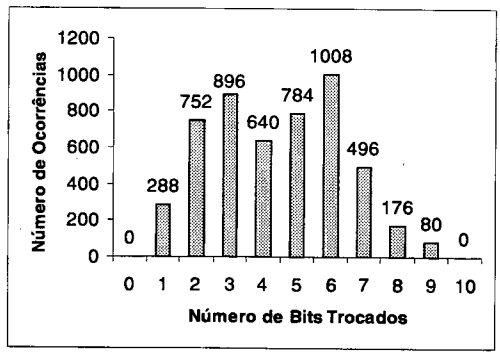


Figura E.45: Reverso de 10 bits - Variando o texto aberto

A figura E.46 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando 10 bits da chave. Neste modelo existem 1168 ocorrências para troca de 4 bits. As maiores ocorrências de troca de bits estão concentradas nos bits 4, 5 e 6. O aumento e diminuição de ocorrências de troca, nos outros bits, apresenta uma distribuição aproximadamente uniforme.

A tabela E.23 mostra a variação da média, desvio padrão e moda calculada para o modelo de 10 bits. Este modelo, variando o texto aberto não apresenta

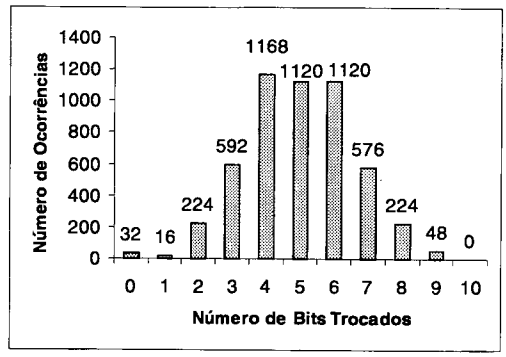


Figura E.46: Reverso de 10 bits - Variando a chave

uma distribuição uniforme de ocorrências de troca. Mas variando a chave, a distribuição de ocorrências de troca, é aproximadamente uniforme. Os valores de desvio padrão são próximos.

Tabela E.23: Dados Analíticos para Modelo de 10 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 4,42 | 1,96 | 6 | 1008 |
| chave | 4,98 | 1,58 | 4 | 1168 |

**Modelo de 11 bits para Criptossistema Reverso**

Para um arquivo de 11 bits, as possibilidades de troca de um bit são **11264**. A figura E.47 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 11 bits. Neste modelo existem 1952 ocorrências para troca de 4 bits. A distribuição de ocorrências de troca é uniforme, exceto quando 5 bits são trocados.

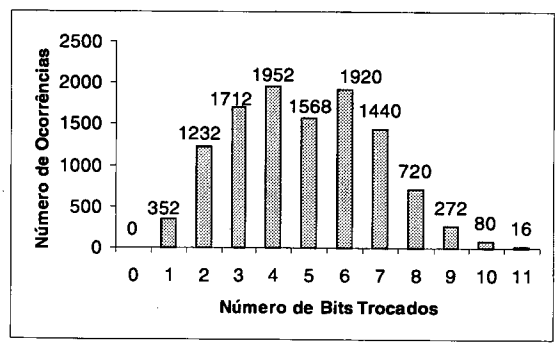


Figura E.47: Reverso de 11 bits - Variando o texto aberto

A figura E.48 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando a chave de 11 bits. Neste modelo existem 2496 ocorrências para troca

de 6 bits. A distribuição de ocorrências apresenta uma distribuição aproximadamente uniforme e o modelo se aproxima a uma curva gaussiana.

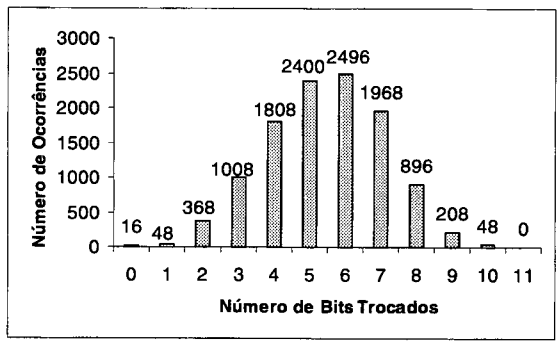


Figura E.48: Reverso de 11 bits - Variando a chave

A tabela E.24 mostra a variação da média, desvio padrão e moda calculada para o modelo de 11 bits. Este modelo apresenta uma distribuição uniforme de ocorrências de troca, exceto quando 5 bits são trocados. Mas variando a chave, a distribuição de ocorrências de troca, é uniforme e o modelo se aproxima a uma curva gaussiana.

Tabela E.24: Dados Analíticos para Modelo de 11 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 4,83 | 2,03 | 4 | 1952 |
| chave | 5,44 | 1,66 | 6 | 2496 |

**Modelo de 12 bits para Criptossistema Reverso**

Para um arquivo de 12 bits, as possibilidades de troca de um bit são **24576**. A figura E.49 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 12 bits. Neste modelo, existem 4352 ocorrências para troca de 4 bits. A distribuição de ocorrências de troca é uniforme, exceto quando 5 bits são trocados.

A figura E.50 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando a chave de 12 bits. Neste modelo existem 5344 ocorrências para troca de 6 bits. A distribuição de ocorrências apresenta uma distribuição uniforme e o modelo se aproxima a uma curva gaussiana.

A tabela E.25 mostra a variação da média, desvio padrão e moda calculada para o modelo de 12 bits. Este modelo, variando o texto aberto, apresenta uma distribuição uniforme de ocorrências de troca, exceto quando 5 bits são trocados. Entretanto, variando a chave, a distribuição de ocorrências de troca, é uniforme e o modelo se aproxima a uma curva gaussiana.

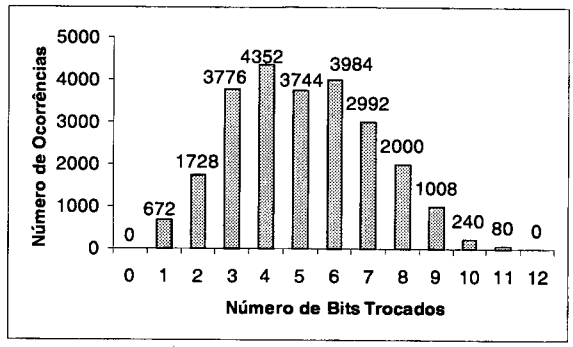


Figura E.49: Reverso de 12 bits - Variando o texto aberto

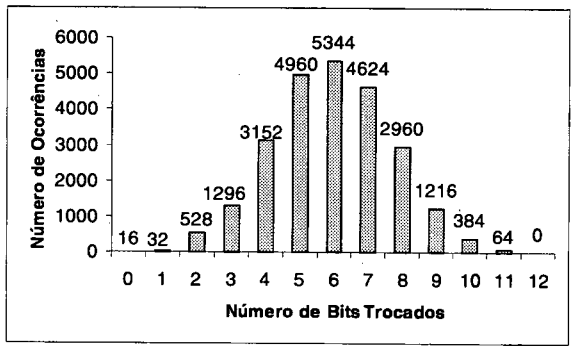


Figura E.50: Reverso de 12 bits - Variando a chave

Tabela E.25: Dados Analíticos para Modelo de 12 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 5,08 | 2,06 | 4 | 4352 |
| chave | 5,94 | 1,74 | 6 | 5344 |

**Modelo de 13 bits para Criptossistema Reverso**

Para um arquivo de 13 bits, as possibilidades de troca de um bit são **53248**. A figura E.51 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 13 bits. Neste modelo existem 9024 ocorrências para troca de 5 bits. A distribuição de ocorrências de troca é uniforme. O modelo se aproxima a uma curva gaussiana.

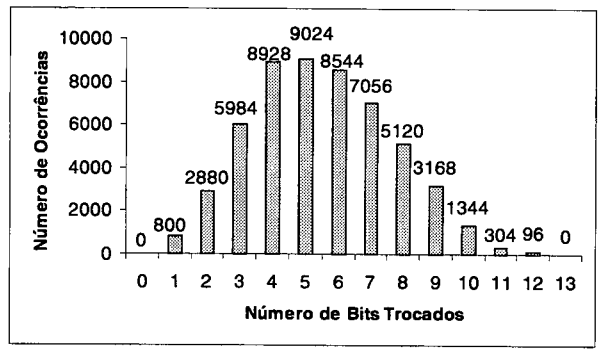


Figura E.51: Reverso de 13 bits - Variando o texto aberto

A figura E.52 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando a chave de 13 bits. Neste modelo existem 10816 ocorrências para troca de 6 bits. A distribuição de ocorrências apresenta uma distribuição uniforme e o modelo se aproxima a uma curva gaussiana.

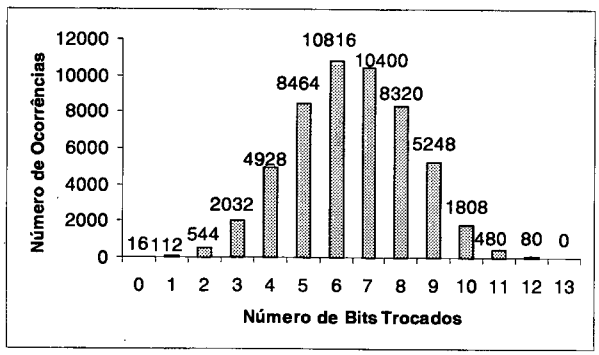


Figura E.52: Reverso de 13 bits - Variando a chave

A tabela E.26 mostra a variação da média, desvio padrão e moda calculada para o modelo de 13 bits. Neste modelo variando o texto aberto ou a chave, apresenta uma distribuição uniforme de ocorrências de troca, aproximando-se a uma curva gaussiana.

Tabela E.26: Dados Analíticos para Modelo de 13 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 5,51 | 2,13 | 5 | 9024 |
| chave | 6,48 | 1,84 | 6 | 10816 |

**Modelo de 14 bits para Criptossistema Reverso**

Para um arquivo de 14 bits, as possibilidades de troca de um bit são **114688**. A figura E.53 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 14 bits. Neste modelo existem 19888 ocorrências para troca de 6 bits. A distribuição de ocorrências de troca é aproximadamente uniforme. O modelo se aproxima a uma curva gaussiana.

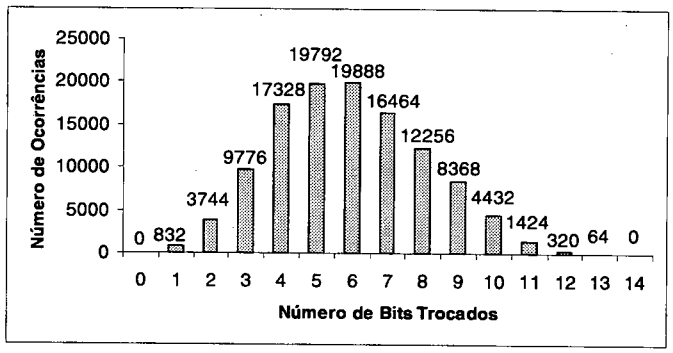


Figura E.53: Reverso de 14 bits - Variando o texto aberto

A figura E.54 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando a chave de 14 bits. Neste modelo existem 23536 ocorrências para troca de 7 bits. A distribuição de ocorrências é uniforme e o modelo se aproxima a uma curva gaussiana.

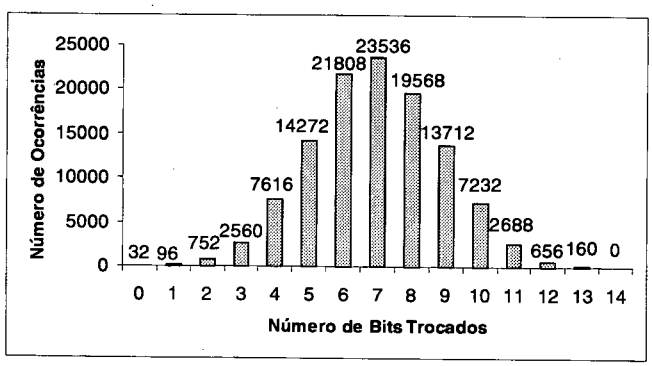


Figura E.54: Reverso de 14 bits - Variando a chave

A tabela E.27 mostra a variação da média, desvio padrão e moda calculada para o modelo de 14 bits. Este modelo, variando o texto aberto ou a chave, apresenta

uma distribuição uniforme de ocorrências de troca, aproximando-se a uma curva gaussiana. Os valores para o desvio padrão são próximos.

Tabela E.27: Dados Analíticos para Modelo de 14 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 5,92 | 2,13 | 6 | 19888 |
| chave | 6,96 | 1,91 | 7 | 23536 |

### Modelo de 15 bits para Criptossistema Reverso

Para um arquivo de 15 bits, as possibilidades de troca de um bit são **245760**. A figura E.55 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando o texto aberto de 15 bits. Neste modelo existem 43072 ocorrências para troca de 6 bits. A distribuição de ocorrências de troca é aproximadamente uniforme. O modelo se aproxima a uma curva gaussiana.

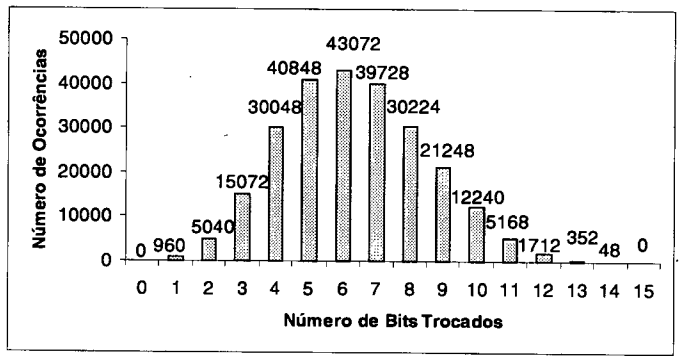


Figura E.55: Reverso de 15 bits - Variando o texto aberto

A figura E.56 ilustra o efeito sobre o texto cifrado com o cifrador reverso, variando a chave de 15 bits. Neste modelo existem 48400 ocorrências para troca de 8 bits. A distribuição de ocorrências é uniforme e o modelo se aproxima a uma curva gaussiana.

A tabela E.28 mostra a variação da média, desvio padrão e moda calculada para o modelo de 15 bits. Este modelo, variando o texto aberto ou a chave, apresenta uma distribuição uniforme de ocorrências de troca, aproximando-se a uma curva gaussiana. Os valores para o desvio padrão são próximos.

Tabela E.28: Dados Analíticos para Modelo de 15 bits do Criptossistema Reverso

| Variação | Média | Desvio Padrão | Moda | Quantidade |
|---|---|---|---|---|
| aberto | 6,33 | 2,15 | 6 | 43072 |
| chave | 7,49 | 1,95 | 8 | 48400 |

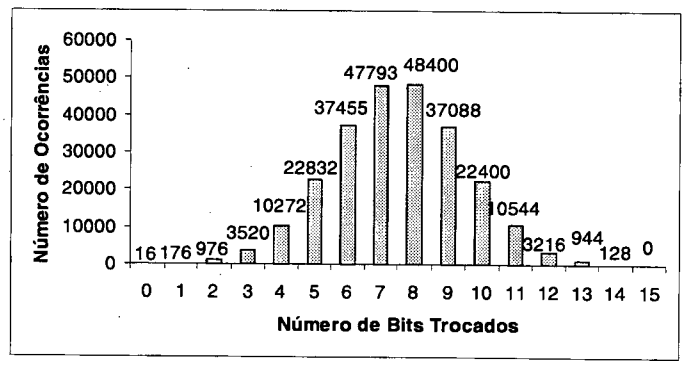


Figura E.56: Reverso de 15 bits - Variando a chave