MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
SECRETARIA DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

DANIEL PORDEUS MENEZES

ADICIONANDO SEGURANÇA AO ALGORTIMO DE COMPRESSÃO *BLOCK-SORTING*

Rio de Janeiro
Abril de 2008

INSTITUTO MILITAR DE ENGENHARIA

DANIEL PORDEUS MENEZES

ADICIONANDO SEGURANÇA AO ALGORTIMO DE COMPRESSÃO *BLOCK-SORTING*

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Prof. Major Claudio Gomes de Mello - DC

Rio de Janeiro
Abril de 2008

cAbril de 2008

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80-Praia Vermelha
Rio de Janeiro-RJ CEP 22290-270

XXXXXMenezes, D. P.
Adicionando Segurança ao Algortimo de Compressão *Block-Sorting*/ Daniel Pordeus Menezes.
– Rio de Janeiro: Instituto Militar de Engenharia, Abril de 2008.
xxx p.: il., tab.

Dissertação (mestrado) – Instituto Militar de Engenharia – Rio de Janeiro, Abril de 2008.

1. Robôs móveis autônomos. 2. Robôs cooperativos. I. Título. II. Instituto Militar de Engenharia.

CDD 629.892

INSTITUTO MILITAR DE ENGENHARIA

DANIEL PORDEUS MENEZES

ADICIONANDO SEGURANÇA AO ALGORTIMO DE COMPRESSÃO *BLOCK-SORTING*

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Prof. Major Claudio Gomes de Mello - DC

Aprovada em xx de xxxxxx de 2008 pela seguinte Banca Examinadora:

Prof. Major Claudio Gomes de Mello - DC do IME - Presidente

Prof. XXXXXX - Ph.D, do XXXX

Prof. YYYYYY - DSc, do YYYY

Rio de Janeiro
Abril de 2008

Dedico esta dissertação a meus pais, Jonas Menezes Pereira e Maria de Nazareth Pordeus Menezes, a minha avó materna Maria Pordeus e a meus avós paternos, que Deus os tenha em sua paz, Valquiria Meneses e Zé Bento Pereira.

AGRADECIMENTOS

Agradeço a todas as pessoas que contribuíram com o desenvolvimento desta dissertação de mestrado, tenha sido por meio de críticas, idéias, apoio, incentivo ou qualquer outra forma de auxílio. Em especial, desejo agradecer às pesssoas citadas a seguir.

Ao meu orientador Major Claudio Gomes de Mello pelo seu apoio, pela sua disponibilidade e acima de tudo, pela confiança em mim depositada, me aceitando como seu orientando, ajudando-me a vencer os desafios que uma dissertação de mestrado apresenta.

Aos professores Xéxeu, Claudia Justel e Claudia Oliveira, que me ensinaram sobre pesquisa cientifica durante todo o primeiro ano como mestrando do IME.

Ao professor Paulo Rosa, que me encorajou a encarar o curso do IME em paralelo com o trabalho.

Ao meu amigo Tiago Macambira, que sempre acreditou em mim desde a época da faculdade e tendo paciência para me explicar os detalhes de programação que parecia só ele saber. Muito de minhas vitórias durante o curso de mestrado do IME devo a ele.

Ao meu primo João Paulo, que me serviu de inspiração para voltar a encarar um mestrado.

Ao meu amigo André Casimiro, por sempre estar disposto a me falar sobre pesquisas cientificas, como guiá-las e documentá-las, sem esquecer de mencionar sua capacidade de companheiro e amizade.

Aos meus amigos Eugênio, João Francisco, Betinho, Brito, Remo e Ana Paula, por serem quem são: verdeiros e companheiros. Amigos de verdade sempre estão ao seu lado nos momentos mais complicados. Estes nunca me abandonaram.

Aos meus irmãos Rafael e Samuel, por serem especiais pra mim em todas as ocasiões.

Aos meus colegas de IME Daniel Gomes, Rafael, Luciano, Viana, Freire e Maurício, que tentaram de todas as formas me ajudar a prosseguir no curso

Aos meus colegas de trabalho, Sérgio e Marcelo, por dividirem comigo o gosto pelos números, criptográfia e pela segurança da informação.

À Mariana e Daniela, que tanta paciência tiveram durante o decorrer deste mestrado.

À Petrobras e aos meus ex e atuais gerentes, Alfred, Márcio e Pedro, pelo apoio incondicional à conclusão deste trabalho.

Por fim, a todos os professores e funcionários do Departamento de Engenharia de Sistemas (SE/8) do Instituto Militar de Engenharia.

Daniel Pordeus Menezes

'O homem tem medo de ir além dos seus limites...'

Nikola Tesla

SUMÁRIO

LISTA DE ILUSTRAÇÕES

RESUMO

Criptografia e compressão de dados são funcionalidades essenciais quando dados digitais são armazenados ou transmitidos através de canais inseguros. Geralmente, duas operações seqëenciais são aplicadas: primeiro, compressão de dados para economizar espaço de armazenamento e reduzir custos de transmissão, segundo, criptografia de dados para prover confidencialidade. Essa solução funciona bem para a maioria das aplicações, mas é necessário executar duas operações de grande esforço computacional. Neste trabalho são propostas modificações no algoritmo de compressão *Block-Sorting* para que este realize compressão e criptografia de dados, simultaneamente.

A contribuição deste trabalho é a criação de um novo algoritmo cripto-compressor, o BZip2s, que é uma versão do BZip2, encontrado em distribuições Linux. O BZip2s apresenta vantagens em relação a outros algoritmos cripto-compressores por obter um resultado melhor que a criptografia de Huffman original, além de possuir diversas possibilidades de configuração, permitindo variar entre uma melhor compressão, uma maior segurança ou um equilíbrio entre ambos. Diversos testes foram realizados, assim como resultados comparativos entre os métodos compressores originais e o BZip2s.

ABSTRACT

Data compression and encryption are essential features when digital data is stored or transmitted over insecure channels. Usually, we apply two sequential operations: first, we apply data compression to save disk space and to reduce transmission costs, and second, data encryption to provide confidentiality. This solution works fine for most applications, but we have to execute two expensive operations. In this work we propose algorithms that achieve both compressed and encrypted data, simultaneously.

The contribution of this thesis is creation of a new crypto-compressor algorithm, the Bzip2s, a new version of BZip2 compressor. BZip2s brings some advantages when compared with others crypto-compressors because BZip2s improves the Huffman result, beyond that, it allows many possibilities of configuration, varying among a better compression, strong security or a balance between than. Some tests was carried through, so the comparisons among original compression methods and the BZip2s results.

1 INTRODUÇÃO

Soluções serializadas, onde primeiro o texto é comprimido e depois cifrado, e então posteriormente decifrado e descomprimido, são atualmente rápidas e eficientes, mas não permitem um real aproveitamento das caracteristicas que um método pode prover ao outro. Algoritmos compressores utilizam confusão e difusão para tentar reduzir a redundância do texto. Na verdade, compressores de dados são algoritmos criptográficos chaveados pelo próprio contexto, ou seja, a chave criptográfica de um compressor é a natureza do texto a ser comprimido.

Uma prova de que métodos compressores são bons para proteger a informação é o trabalho apresentado em (44), onde Rivest *et al.* (44) fizeram a criptoanálise de uma mensagem codificada com Huffman na qual o criptoanalista não conhece o modelo (alfabeto de símbolos). O referido trabalho também demonstrou que a criptoanálise é extremamente difícil e até mesmo impossível em alguns casos, dado que a mensagem pode ser ambígua, ou seja, pode ter sido gerada por dois códigos de Huffman igualmente válidos.

A cifragem e a compressão são requisitos essenciais para a transmissão e armazenamento de dados em meios digitais. A transferência digital de dados é um serviço amplamente utilizado atualmente, e o custo da utilização da rede para a transferência é proporcional à quantidade dos dados enviados. Logo, para a diminuição do volume de dados transferidos, são utilizadas técnicas de compressão. E, para garantir a segurança de um canal de dados ou de uma mídia digital de armazenamento, são aplicadas técnicas criptográficas.

Nesta dissertação é proposto um algoritmo que faz a compressão e a cifragem simultaneamente, o BZip2s. Usualmente, o arquivo a ser transmitido passa primeiro pelo processo de compressão de dados, e a seguir é criptografado. Este processamento é feito de forma seqüencial, onde o custo total é a soma dos custos individuais da compressão e da criptografia. No algoritmo proposto neste trabalho, o custo total combinado é menor que a soma dos custos convencionais individuais.

O algoritmo é baseado na técnica *Move-to-Front*, etapa intermediária no algoritmo de compressão via ordenação de blocos, o *Block-Sorting Compression*, dividido em Transformada de Burrows-Wheeler, Heurística *Move-to-Front* e compressão de dados. O *Block-*

Sorting foi escolhido por ser um método relativamente novo (foi apresentado em 1994 (15)), por apresentar excelentes resultados de compressão e por não haver, até a data de conclusão desta dissertação, trabalhos relacionados propondo modificações para adicionar segurança.

As alterações apresentadas neste trabalho visam manter a natureza proposta do *Block-Sorting*: melhorar o resultado de métodos compressores como o Huffman, alterando a estatística dos caracteres do texto original. Ou seja, neste trabalho busca-se não degradar o resultado original do *Block-Sorting* a ponto de piorar o resultado final ao invés de melhorá-lo. Para demonstrar o alcance dos objetivos, são apresentados experimentos práticos para avaliar o desempenho em relação às taxas de compressão, além de apresentar análises teóricas e experimentos práticos para avaliar o sigilo dos métodos propostos.

É importante frisar que a proposta não é criar uma algoritmo criptográfico comparável a sistemas já consagrados, como o DES ou o AES, padrões americanos de segurança, que são voltados à privacidade total dos dados e foram desenvolvidos exclusivamente para este fim. O algoritmo aqui proposto é um compressor de dados com o adicional de segurança, visando facilitar a vida dos usuários comuns, permitindo que estes possam preservar dados pessoais em seus computadores domésticos, apenas informando uma senha no momento da compressão de suas informações, buscando equilibrar a balança comodidade x segurança.

O foco das aplicações que se beneficiam deste algoritmo são o armazenamento e distribuição eletrônica segura de coleções textuais. As coleções são podem ser armazenadas em servidores e distribuídas em meio digital (ex: CD, DVD, Pendrive) ou remotamente. O usuário autorizado a acessar a informação recebe uma senha de decodificação através de um meio seguro, para poder ter acesso aos dados locais ou remotos codificados. Os testes realizados mostraram que a utilização de técnicas criptográficas não prejudicaram significativamente as taxas de compressão.

Um exemplo real e prático da aplicação desta proposta é o transporte físico de informações com tempo de vida útil muito curto, como o documento de proposta para uma licitação, numa mídia digital, seja esse um cd ou *pendrive*. Apesar de ser uma informação crítica, por seu tempo de vida ser limitado, já que a proposta só precisa ser protegida até ser entregue ao licitador, não é necessário comprar uma aplicação que proveja criptografia voltada para sigilo abslouto, que poderia ter um custo elevado. Além disso, um documento de proposta de licitação pode conter várias páginas, com comprovações de conhecimento técnico presente na empresa, aumentando o tamanho do documento, tornando interes-

sante aplicar um compressor de dados sobre o documento. Utilizando o sistema proposto nesta dissertação, um concorrente que venha a se apoderar do documento não conseguirá decodificar a informação num tempo que lhe seja útil para fazer uma proposta com preço menor ao licitador.

Esta dissertação está organizada da seguinte maneira: No capítulo 2 são definidos alguns conceitos sobre teoria da informação, tais como entropia, redundância e distância de Hamming.

No capítulo 3 são apresentados conceitos sobre algoritmos criptográficos, segurança de sistemas e tipos de ataque.

Os capítulos 4 e 5 são sobre sistemas compressores. O capítulo 4 define o que são estes sistemas, apresenta métodos de classificação e descreve resumidamente o funcionamento de alguns dos mais conhecidos métodos compressores encontrados na literatura. O capítulo 5 é voltado exclusivamente para a apresentação do Algoritmo de Compressão *Block-Sorting*, descrevendo suas etapas, tanto de codificação como de decodificação/reversão.

O capítulo 6 traz a definição do termo 'cripto-compressor', apresentando os trabalhos publicados que são relacionados ao tema desta dissertação, incluindo uma descrição sucinta de todos os métodos encontrados na literatura.

Os capítulos 7 e 8 apresentam as contribuições desta dissertação, o que foi realizado. O capítulo 7 contém as alterações propostas e realizadas no *Block-Sorting* para prover-lhe segurança. O capítulo 8 trás uma descrição dos testes aplicados ao sistema proposto, seus resultados e gráficos e tabelas comparativas, além de um estudo sobre a segurança da chave criptográfica.

Por fim, no capítulo 9 são apresentadas as conclusões deste trabalho e possíveis continuações desta pesquisa.

2 TEORIA DA INFORMAÇÃO

2.1 INTRODUÇÃO

A Teoria da Informação foi apresentada em 1948 por Shannon (7) e tem grande importância no estudo da criptografia, transmissão de informação e compressão de dados.

As palavras **incerteza**, **informação** e **redundância** tem um certo intuitivismo em seu significado. O termo **entropia** da termodinâmica pode sugerir uma noção relacionada, conhecida por *grau de desordem*. Isso pode ser mais preciso e para o contexto deste capítulo, assume-se que os três termos - incerteza, informação e entropia - referem-se aproximadamente à mesma coisa, enquanto redundância refere-se à falta de incerteza (3).

A idéia intuitiva de incerteza em algum experimento (tal como jogar uma moeda ou rolar dados) mede o quanto de informação se pode obter depois que o experimento foi concluído. Entropia é o sinônimo de incerteza neste sentido.

Este capítulo apresenta alguns conceitos básicos sobre Teoria da Informação que serão utilizados durante este trabalho.

2.2 FONTE DE INFORMAÇÃO

Uma **fonte de informação** é o que emite, produz ou apresenta informações através de símbolos pertencentes a um conjunto finito, o **alfabeto**. Uma fonte emite mensagens na forma de textos, mas quanto de informação pode ser produzida por uma fonte? Esse conhecimento é necessário para utilizar o conhecimento estatístico sobre a fonte com intuíto de reduzir a geração de informação a fim de se adequar à capacidade do canal de transmissão.

2.3 ALFABETO

Um alfabeto $\Sigma = (\sigma_1, \ldots, \sigma_n)$ é um conjunto finito composto de n símbolos distintos σ_i, com $i \in [1, n]$, denominados letras do alfabeto.

Um texto T é uma concatenação de m caracteres $t_1 \ldots t_m$, não necessariamente distintos, onde cada caractere é uma letra do alfabeto ($t_i \in \Sigma$, com $i \in [1, m]$).

Por exemplo, o texto 'MENEZES' possui 7 caracteres e 5 letras, com o alfabeto $\Sigma = (' E',' M',' N',' S',' Z')$.

2.4 ENTROPIA

Informação e incerteza descrevem qualquer processo que seleciona um ou mais objetos de um conjunto (28) . Por exemplo, suponha que uma fonte que pode emitir três mensagens A, B ou C. A incerteza se caracteriza por não se saber qual será o próximo símbolo que será produzido. Quando a mensagem aparece, essa incerteza desaparece e pode-se dizer que se adquire uma informação. Assim, um acréscimo de informação resulta em uma diminuição da incerteza (35; 28), reduzindo a entropia do sistema.

A **entropia** de uma fonte de informação é a medida da quantidade de informação que a fonte possui e pode ser medida matematicamente como uma função de distribuição probabilística sobre o conjunto de todas as mensagens possíveis que a fonte pode produzir.

Supondo uma variável aleatória $X = (x_1, x_2, ..., x_n)$ ocorrendo de acordo com uma distribuição de probabilidade $p(X)$, a entropia de X, é definida formalmente por $H(X)$, onde

$$H(X) = -\sum_{i=1}^{n} p(x_i) * \log_2 p(x_i)$$

O conceito geral de entropia de uma fonte (8) é a quantidade média de informação contida na fonte (35). Para uma fonte $\Gamma = (T_1, T_2, \ldots)$, a entropia $H(\Gamma)$ é a média ponderada de todos os símbolos dessa fonte, com pesos iguais às suas probabilidades de ocorrência

$$H(\Gamma) = -\sum_{i=1}^{n} p(T_i) * \log_2 p(T_i).$$

O cálculo da entropia do texto é feita a partir das frequências de ocorrência f_i das letras no texto

$$H(T) = -\sum_{i=1}^{n} p(t_i) * \log_2 p(t_i), \text{ onde } p(t_i) = \tfrac{1}{f_i}.$$

A parcela $-\log^p(t_i)_2$ é chamada de quantidade de informação de t_i expressa em bits. Então, a entropia é a soma da probabilidade de cada letra multiplicada pela sua respectiva quantidade de informação. Podemos entender a entropia como sendo a quantidade de informação média, ou seja, um limite inferior para o comprimento médio dos códigos relativos a cada instância gerada pela fonte.

Por exemplo, seja T um texto que contém 3 letras, σ_1, σ_2 *e* σ_3, com probabilidades de ocorrência no texto iguais a $\frac{1}{2}$, $\frac{1}{4}$ e $\frac{1}{4}$, respectivamente. Então o número médio de bits para codificar T é igual a

$$H(T) = -(\tfrac{1}{2}.\log_2^{\frac{1}{2}} + \tfrac{1}{4}.\log_2^{\frac{1}{4}} + \tfrac{1}{4}.\log_2^{\frac{1}{4}}) = 1,5\ bits/letra.$$

2.4.0.1 EXEMPLOS DE UTILIZAÇÃO DA EQUAÇÃO DE ENTROPIA

A seguir são apresentados exemplos da utilização da equação de entropia para facilitar o entendimento. Estes exemplos foram retirados de (28)

- **Exemplo 1:** Suponha que X representa o lançamento de uma moeda honesta. A distribuição de probabilidades é $p(cara) = p(coroa) = 1/2$.
  $H(T) = -(\frac{1}{2}.\log_2^{\frac{1}{2}} + \frac{1}{2}.\log_2^{\frac{1}{2}}) = 1$. O resultado mostra que é possível representar o lançamento de cara ou coroa de uma moeda utilizando apenas 1 bit: valor 1 representa cara, 0 coroa.

- **Exemplo 2:** Suponha agora que seja necessário codificar os dias da semana e que todos possuem a mesma probabilidade de ocorrência, permitindo utilizar a equação de uma maneira simplificada:
  $H(T) = \log_2^7 = 2,8$. Esse resultado mostra que não é preciso utilizar todas as combinações de 3 bits. A tabela 2.1 traz uma possível resposta para este mapeamento.

Código	Símbolo
000	Domingo
001	Segunda
010	Terça
011	Quarta
100	Quinta
101	Sexta
110	Sábado

TAB. 2.1: Dias da Semana

- **Exemplo 3:** A tabela 2.2 apresenta a frequência média de ocorrência dos caracteres da lingua portuguesa. Qual a quantidade de bits para codificar as letras deste alfabeto?

  Calculando a entropia da lingua portuguesa, tem-se:

Letra	Frequência	Letra	Frequência
A	14.64%	N	4.85%
B	1.16%	O	10.8%
C	3.76%	P	2.58%
D	4.97%	Q	1.09%
E	12.7%	R	6.88%
F	1.02%	S	7.97%
G	1.29%	T	4.26%
H	1.42%	U	4.42%
I	5.90%	V	1.68%
J	0.32%	W	0.01%
K	0.01%	X	0.23%
L	2.95%	Y	0.01%
M	4.71%	Z	0.42%

TAB. 2.2: Ocorrência de Caracteres na Lingua Portuguêsa

$$H(T) = -(p('A').\log^p('A')_2 + \ldots + p('Z').\log^p('Z')_2) = 1,25279 bits/letra$$

As propriedades da entropia são:

- **Propriedade 1:** $0 \leq H(X) \leq \log_2^n$;
- **Propriedade 2:** $H(X) = 0 \iff p(x_i) = 1$ para algum $i, 1 \leq i \leq n$, e $p(x_j) = 0$ para todo $j, i \neq j, 1 \leq j \leq n$;
- **Propriedade 3:** $H(X) = \log_2^n \iff p(x_i) = 1/n$ para cada $i, 1 \leq i \leq n$.

Em um sistema criptográfico, pode-se calcular a entropia dos vários componentes. Pode-se pensar na chave como sendo uma variável aleatória K que assume valores de acordo com a distribuição de probabilidade p(k) e, consequentemente, pode-se calcular a entropia H(K). Analogamente, podemos calcular as entropias H(T) e H(C) associadas com as distribuições de probabilidade do texto original e do texto cifrado.

Quanto maior a entropia das chaves H(K) e dos textos cifrados H(C), maior será o nível de incerteza para o criptoanalista, pois o procedimento de se capturar um texto cifrado e, através de técnica de força bruta, tenta-se encontrar o texto original torna-se inviável.

2.4.1 REDUNDÂNCIA EM LINGUAGENS NATURAIS

Este tópico tem como referência o cápitulo 3 do livro (2).

Considere que a fonte S(L) produza palavras na liguagem natural L. Suponha que, em média, cada palavra em L tem k caracteres. Desde o teorema de Shanon, H(S(L)) é a média mínima do número de bits por palavra produzida por S(L), então o valor

$$r(L) = \frac{H(S(L))}{k}$$

seria a média miníma do número de bits por caracter na liguagem L. O valor r(L) é chama ***taxa da liguagem*** *L. Seja L a lingua inglesa. Shanon calculou que r(Inglês) é um valor entre 1 e 1,5 bits por letra.*

Seja $\Sigma = \{a, b, .., z\}$. *Então sabe-se que* $r(\Sigma) = \log_2(26) \approx 4,7$ *bits/letra.* $r(\Sigma)$ *é chamada* ***taxa absoluta da linguagem*** *com o conjunto-alfabeto* Σ. *Comparando r(Inglês) com* $r(\Sigma)$. *vê-se que a atual taxa da lingua inglesa é considerada menor que a taxa absoluta.*

A **redundância da liguagem** L com o conjunto-alfabeto Σ é
$r(\Sigma) - r(L)$ bits por caractere.

Para uma consideração conservativa de r(Inglês) = 1,5, a redundância desta língua é $4,7 - 1,5 = 3,2$ *bits por letra. Em termos percentuais, a taxa de redundância é* $3,2/4,7 \approx 68\%$ *Em outras palavras, cerca de 68% de letras das palavras do Inglês são redundantes. Isto significa uma possibilidade de comprimir um artigo em inglês para 32% do volume original sem perda de informação.*

Redundância em linguagens naturais surge de conhecido e freqüente aparecimento de padrões em uma linguagem. Por exemplo, no Inglês, a letra q é quase sempre seguida de u; 'the', 'ing' e 'ed' são outros exemplos de padrões conhecidos. Redundância em linguagens naturais provê um importante significado para a ***criptanálise****, que busca recuperar as mensagens originais ou a chave criptográfica a partir do texto cifrado.*

2.5 CONFUSÃO E DIFUSÃO

Confusão e difusão são técnicas descritas por Shannon (7; 8) para obscurecer as redundâncias de uma mensagem, frustando análises estatisticas. São técnicas básicas de criptografia utilizadas para modificar a frequência e a posição originais dos caracteres, eliminando possíveis atalhos que poderiam revelar alguma informação sobre o texto claro somente analisando-se o texto cifrado.

Pode-se dizer que uma boa cifra de substituição acrescenta confusão à informação, enquanto uma boa cifra de transposição acrescenta difusão. A criptografia moderna, que será apresentada no capítulo 3, item 3.2.4 deste trabalho, efetua diversas operações com essas caracteristicas repetidas vezes.

Definições encontradas em (9):

- **Confusão:** tem a propriedade de esconder a relação entre o texto original, o texto cifrado e a chave, evitando que redundâncias e padrões estatísticos permitam adivinhar algo sobre o texto claro analisando o texto cifrado. O modo mais fácil para se fazer isto é utilizar a substituição. Uma cifra de substituição simples, como a Cifra de César, não é a forma mais adequada, porque ela substitui uma letra do texto original por outra no texto cifrado. As cifras de substituição modernas são mais complexas. Nelas, um bloco do texto original é substituído por um bloco diferente no texto cifrado e as regras da substituição mudam constantemente, normalmente misturando informações de outros blocos. Uma boa confusão torna o relacionamento estatistíco são complicado que até mesmo uma ferramenta criptoanalítica poderosa não funcionaria. O método de confusão por si só já é suficiente para segurança.

- **Difusão:** tem a propriedade de dissipar a redundância do texto original de forma que as redundâncias fiquem espalhadas no texto cifrado. O criptoanalista terá mais dificuldades para encontrar algum padrão que o ajude na decodificação. O modo mais simples para causar difusão é realizando transposições. Uma cifra de transposição simples, como transposição de colunas, apenas reorganiza as letras do texto original. Cifras modernas realizam não apenas esta, mas também executam outras formas de difusão combinadas permitindo dissipar partes do texto por toda a mensagem.

2.6 PESO E DISTÂNCIA DE HAMMING

Este tópico é baseado no capítulo 13 de (3).

Sejam $v = (v_1, \ldots, v_n)$ e $w = (w_1, \ldots, w_n)$ dois vetores de tamanho n. O **peso de Hamming**, $H_w(v)$ é a o número de entradas v_i que não são 0. A **distância de Hamming** $H_d(v, w)$ entre os dois vetores é o número de posições em que estes diferem. Ou seja, a *distância de Hamming* entre v e w é o *peso de Hamming* da diferença

$$H_d(v, w) = H_w(v - w) \Rightarrow H_d(v, w) = (v_1 - w_1, v_2 - w_2, \ldots, v_n = w_n)$$

A distância de Hamming também pode ser escrita de outra forma simplificada:

$$H_d(v, w) = \text{número de indices } i \text{ tal que } v_i \neq w_i$$

Em outras palavras, $H_d(v, w)$ é a quantidade de valores que necessitam ser alterados para que o vetor v seja igual a w. Por exemplo, sejam s_1 e s_2 as sequências '00100100' e '10100101', respectivamente. A $H_d(s_1, s_2) = 2$, já que apenas o primeiro e o último bit da cadeia diferem.

A distância de Hamming foi originalmente criada para utilização entre vetores binários, mas é facilmente estendida a vetores de outra natureza. Além da utilização em testes estatísticos, ela também é aplicada em detecção e correção de erros de códigos, transmissão de mensagens, etc.

3 SISTEMAS CRIPTOGRÁFICOS

3.1 INTRODUÇÃO

Criptografia é uma das principais ferramentas para prover segurança em meios eletrônicos. A criptografia compreende os princípios, meios e métodos para garantir a integridade, confidencialidade e autenticidade da informação. De forma alguma ela é a única ferramenta necessária para a segurança dos dados e também não resolverá todos os problemas relativos a segurança dos dados até por não ser infalível, mas sua ausência num sistema de informações sensíveis é praticamente inaceitável.

Um sistema criptográfico tem como objetivo permitir que duas pessoas possam se comunicar em um canal inseguro, e deve assegurar que um indivíduo não autorizado não consiga entender o que está sendo transmitido. Para isso, o sistema deverá utilizar algum tipo de criptografia para codificar a mensagem, transmití-la cifrada e decifrá-la no destino.

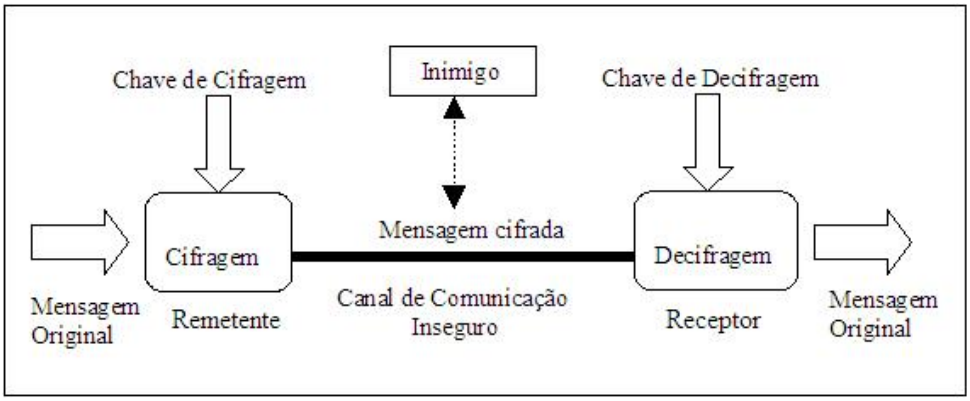


FIG. 3.1: Sistema Criptográfico

A listagem abaixo fornece alguns dos serviços gerais requeridos em um ambiente de segurança da informação que são foco de pesquisas e desenvolvimento em segurança de redes e computadores (4):

- **Integridade:** A garantia de que a informação não foi modificada desde o momento de sua concepção até o momento da leitura por quem ela foi destinada. A integridade garante que somente pessoas autorizadas podem modificar (escrever, alterar, deletar, criar) o ativo eletrônico;

- **Confidencialidade:** A informação deve ser acessada apenas por quem possui autorização. Garante que a informação num sistema computacional ou numa transmissão são acessíveis para leitura (impressão, exibição e outras formas de exposição de conteúdo) apenas por pessoas autorizadas;
- **Autenticidade:** Garante que a origem da informação é corretamente identificada, com a garantia de que essa identidade não é falsa;
- **Disponibilidade:** A informação deve estar disponível a quem tem autorização de acesso no momento em que ela é necessária;
- **Não-Repúdio:** Garantia de não-negação por parte do criador e do receptor da informação;
- **Controle de Acesso:** Requer que o acesso à informação seja controlado por um sistema com habilidade de limitar e controlar o alcance de um indivíduo numa aplicação ou num link de comunicação.

Estes princípios listados acima são alguns dos ganhos que se obtém com a utilização da criptografia. Os sistemas criptográficos utilizam métodos criptográficos para prover um ou mais dos itens da lista acima. Nas próximas seções serão detalhados alguns destes sistemas.

3.2 CONCEITOS INICIAIS

3.2.1 GLOSSÁRIO DE CRIPTOGRAFIA

Os conceitos que serão apresentados a seguir são formalizados em diversos exemplos na literatura.

- **Texto Claro:** Texto original;
- **Texto Cifrado:** Texto codificado por um sistema criptográfico;
- **Criptografar, Codificar, Cifrar ou Encriptar:** Transformar um texto claro em texto cifrado através de uma função inversível e de uma chave ou senha. $\mathbf{T} = \mathbf{E}_k(\mathrm{C})$;
- **Decriptografar, Decodificar, Decifrar ou Decriptar:** Recuperar o texto claro através da função inversa utilizada na etapa de criptografar. $\mathbf{C} = \mathbf{D}_l(\mathrm{T})$;

- **Chave:** Informação utilizada na operação de cifragem e/ou decifragem. Ninguém que não seja autorizado pode ser acesso a essa informação ou o sistema criptográfico será inútil. Não é necessário que as chaves utilizadas na cifragem e decifragem sejam iguais, mas devem estar matematicamente relacionadas;

- **Canal:** Meio pelo qual a mensagem trafega, sendo este seguro ou não;

- **Força Criptográfica:** Quão resistente a um ataque é a criptografia ou a chave criptográfica;

- **Sistema Criptográfico:** Conjunto formado por um ou mais algoritmos criptográficos, uma coleção de textos claros, textos cifrados e de chaves;

- **Sistema Criptográfico Simétrico:** Sistema criptográfico onde a chave utilizada para cifrar a informação é a mesma utilizada para sua decifragem. Neste caso, somente o remetente e o destinatário da informação devem ter conhecimento do valor da chave, que é conhecida como **Chave Simétrica ou Secreta**;

- **Sistema Criptográfico Assimétrico:** Sistema onde a chave utilizada para cifrar é diferente da chave utilizada para decifrar, mas existe uma relação matemática entre elas. Esta relação deve ser de uma complexidade tal que inviabilize descobrir uma chave através da outra. Neste sistema as chaves são denominadas **Chave Privada** e **Chave Pública**;

- **Criptoanálise:** É a ciência de quebrar códigos, decodificar segredos, violar esquemas de autenticação e quebrar protocolos criptográficos e compreende métodos de atacar as fraquezas dos algoritmos criptográficos para obter o texto claro através do texto cifrado sem o conhecimento da chave ou mesmo a tentativa de descobrir a chave através da análise do texto cifrado;

- **Criptoanalista:** Pessoa interessada em descobrir a informação criptografada; pessoa que faz criptoanálise;

- **Ataque Estatístico:** Método criptoanalitíco baseado em contagem de caracteres, buscando comparar a estatística do texto cifrado com a de uma língua natural, como o Português. Funciona muito bem com cifras monoalfabéticas.

3.2.2 CRIPTOLOGIA

A palavra criptografia é de origem grega e significa 'escrita secreta'. Criptologia é a ciência que reúne o estudo e os métodos e técnicas para escrever (comunicar-se) secretamente, assim como também as técnicas e os conhecimentos para realizar a reversão desta escrita, conhecido como criptoanálise. Há indícios de utilização desta arte pelos egípcios desde 1900 a.C.

Algoritmos criptográficos são concebidos para 'esconder' informações sigilosas de qualquer pessoa desautorizada a lê-las. Um sistema criptográfico é a implementação destes algoritmos e possui os seguintes componentes básicos (23):

- Um espaço **T** de textos em claro;
- Um espaço **C** de textos cifrados, ou criptogramas;
- Um espaço **K** de chaves;
- Uma família $\mathbf{E}_k$ de Transformações $\mathbf{C} \rightarrow \mathbf{T}$ de cifragem, com k $\in \mathbf{K}$;
- Uma família $\mathbf{D}_l$ de Transformações $\mathbf{T} \rightarrow \mathbf{C}$ de decifragem, com l $\in \mathbf{K}$.

Para todas as chaves k,l $\in \mathbf{K}$, $\mathbf{D}_l$ é a inversa de $\mathbf{E}_k$. Logo, para t $\in \mathbf{T}$ temos:

$$\mathbf{D}_l\ (\mathbf{E}_k(\mathrm{t})) = \mathrm{t}$$

3.2.3 SEGURANÇA DE SISTEMAS CRIPTOGRÁFICOS

A Criptografia Moderna não é mais baseada na obscuridade, ou seja, não está baseada em algoritmos fechados, secretos, utilizados internamente nos sistemas. Atualmente, a segurança de um sistema criptográfico é baseada na chave. Este mecanismo deve ser tão bom que nem mesmo os desenvolvedores devem ser capazes de decodificar a mensagem sem conhecer essa chave. Desta forma, os criptoanalistas conhecem todo o sistema criptográfico que desejam quebrar, menos a chave que foi utilizada pra codificar a mensagem (28).

Até o momento, não se tem conhecimento de um método objetivo para provar que um algoritmo de criptografia é seguro. A melhor maneira conhecida é publicá-lo e deixar que especialistas tratem de estudá-lo e apresentar suas forças e fraquezas. Podem passar anos até que uma falha apareça, mas ainda assim é o melhor meio de ganhar a confiança das pessoas de segurança da informação.

Outras opções são análises que podem ser feitas durante o desenvolvimento do algoritmo. Uma delas é a comparação da segurança do sistema a problemas computacionalmente difíceis (Problemas NP-Difíceis ou NP-Completos) através de redução matemática. É válido informar que este tipo de prova é apenas relativa, não uma prova absoluta de segurança (35).

Uma outra maneira é baseada em estatística, onde se procura mostrar que o texto cifrado gerado tem um comportamento similar a um texto aleatório. Esse é o princípio da análise proposta pelo *National Insitute of Standards and Technology* (NIST). Estas análises serão detalhadas no capítulo 8, item 8.3 deste trabalho.

Pode-se classificar os sistemas criptográficos em relação a sua segurança de duas maneiras:

- **Segurança Perfeita:** Considere um algoritmo **F** cuja cardinalidade de **T**, **K** e **C** são iguais. Portanto, **F** possui segurança perfeita se e somente se:

  a) Dado um par (x, y), onde $x \in T$, $y \in C$, só existe uma chave $k \in K$ que transforma x em y;

  b) Todas as chaves são equiprováveis.

  A prova da afirmação acima pode ser encontrada em (6).

  Pode-se dar outras definições mais compreensíveis para a segurança perfeita. Em (35), um sistema criptográfico tem segurança incondicional ou perfeita quando não pode ser quebrado, até mesmo com recursos computacionais infinitos. Shannon (8) afirmou que em um sistema criptográfico perfeito, um criptograma não pode fornecer qualquer informação probabilística sobre o texto original que o gerou. Essa definição gerou um modelo matemático preciso, sendo expresso por

  $$p(x/y) = p(x)$$

  onde $p(x)$ é a probabilidade de texto claro original ser x e $p(x/y)$ é a probabilidade condicional de o texto claro ser x dado que o texto codificado é y. Em outras palavras, a cardinalidade de **K** deve ser tão grande quanto a cardinalidade de **T**, como já foi dito, fazendo com que o tamanho da chave utilizada tenha pelo menos o mesmo tamanho da mensagem a ser codificada, além de ser necessário que nenhuma chave já utilizada seja utilizada novamente em outra codificação.

- **Computacionalmente Seguro:** Pode-se dizer que um sistema é computacionalmente seguro se o algoritmo mais eficiente para efetuar uma criptoanálise sobre este sistema necessita de um número grande de operações a ponto de tornar inviável ou pelo menos desvantajoso efetuar estas operações. Desvantajoso aqui refere-se ao custo de obter a informação em relação ao benefício que esta informação trará ao criptoanalista. Este custo pode se referir ao valor financeiro da informação versus o valor gasto para decodificá-la, ou mesmo o tempo gasto para isso. A mensagem pode ter vida útil de 2 dias, mas o criptoanalista gastaria 5 para obtê-la. De nada lhe adiantaria.

3.2.4 CRIPTOGRAFIA CLÁSSICA

3.2.4.1 INTRODUÇÃO

Antes dos computadores, a criptografia era baseada em substituição de um caracter por outro ou trocando (permutando) os caracteres de posição. Os melhores algoritmos realizavam ambas operações, e de preferência várias vezes. Atualmente a complexidade aumentou e, ao invés de caracteres, trabalha-se com bits, mas os métodos básicos continuam a ser utilizados. A maioria dos algoritmos contemporaneos de criptografia ainda combinam elementos de substituição e transposição (28).

3.2.4.2 MONOALFABÉTICOS

As cifras monoalfabéticas utilizam, como o próprio nome diz, apenas um alfabeto. Cada letra ou caractere a ser codificado é transformado em um outro, de maneira fixa. Essa caracteristica é evidente quando falamos na **Cifra de César**, uma das mais antigas e conhecidas cifras monoalfabéticas. Julius César utilizava um esquema de deslocamento das letras do alfabeto para enviar mensagens cifradas aos seus generais durante as batalhas. A codificação se dava deslocando o alfabeto em três posições. Vide tabela 3.1.

Alfabeto Original	A	B	C	D	E	F	G	H	I	J	...
Alfabeto de César	D	E	F	G	H	I	J	K	L	M	...

TAB. 3.1: Cifra de César

Para reverter, basta substituir o caractere pelo que está três posições atrás. Este tipo de comunicação foi muito eficiente durante a época que foi utilizada, já que não havia a cultura de codificar mensagens e portanto, não havia idéia do que significava uma mensagem quando um exército adversário a interceptava. Além disso, a codificação e decodificação eram bastante objetivas e rápidas. Depois que os povos adquiriram tal cultura, a Cifra de César foi facilmente quebrada e abandonada.

Uma outra cifra monoalfabética bastante conhecida é a Cifra do Bastão de Licurgo. Um pedaço de papiro era enrolado ao redor de um bastão e a mensagem era escrita no sentido perpendicular ao papiro enrolado. A chave dessa cifra era o diâmetro do bastão. Os generais do mesmo exército precisavam possuir um bastão com as mesmas dimensões diametrias. Vide imagem 3.2.

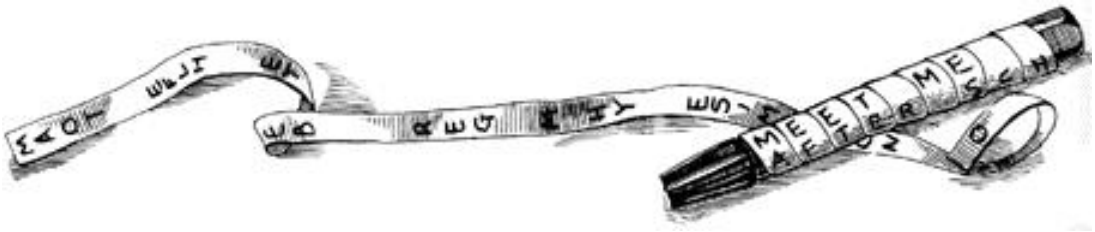


FIG. 3.2: Bastão de Licurgo

Outras cifras monoalfabéticas são definidas facilmente, definindo alfabetos com posições permutadas, sem uma chave. Nesse caso, para decifrar, é preciso que o receptor da mensagem tenha uma cópia do alfabeto codificador. Vide tabela 3.2.

Alfabeto Original	A	B	C	D	E	F	G	H	I	J	...
Alfabeto de Codificação	Z	I	L	A	H	T	J	X	S	O	...

TAB. 3.2: Cifra Monoalfabética

Esses algoritmos de codificação são facilmente quebrados por análise de freqüência dos caracteres. É sabido que em cada língua, umas letras são mais utilizada que

outras. Por exemplo, no alfabeto brasileiro, a letra mais utilizada é a 'A', no inglês, a 'E'. Dessa maneira, fazendo uma contagem dos caracteres numa mensagem relativamente grande e codificada com uma destas cifragens supracitadas, pode-se encontrar que a letra 'X' aparece frequentemente. Se sabe-se que o texto original é de lingua portuguesa, substitui-se o 'X' por 'A' e assim por diante, até decodificar a mensagem por inteiro. Outras técnicas serão mais detalhadas adiante.

3.2.4.3 POLIALFABÉTICOS

Uma cifra de codificação polialfabética é constituída de várias monoalfabéticas. Construindo-se algumas variações do alfabeto original e utilizando cada uma dessas variações de acordo com uma chave tem-se uma codificação polialbfabética. Vide tabela 3.3.

Alfabeto Original	A	B	C	D	E	F	G	H	I	J	...
Alfabeto de Codificação 1	Z	I	L	A	H	T	J	X	S	O	...
Alfabeto de Codificação 2	C	F	B	Y	R	G	K	S	O	X	...
Alfabeto de Codificação 3	D	J	V	G	K	E	S	W	N	U	...

TAB. 3.3: Cifra Polialfabética

Um caso clássico de codificação polialfabética foi a Cifra de Vigenère (10), criada no século XVI. Essa cifra constava de 26 alfabetos com os caracteres deslocados uma posição em relação ao alfabeto imediatamente anterior. A chave definia a ordem de utilização destes alfabetos baseado na primeira coluna de cada alfabeto no momento da subsituição de um dado caractere no texto original. Essa cifra foi considerada inquebrável até que Charles Babbage, em 1854, criou um método de criptoanálise para decodificar Vigenère, buscando padrões dentro de um texto para se descobrir o tamanho da chave e a partir daí aplicar análise de freqüência, porém, este resultado só foi divulgado em 1863, quando Friedrich Kasiski publicou um resultado similar.

Para mostrar o funcionamento desta cifra, tome a palavra 'MESTRE' a ser codificada e a palavra 'IME' para servir como chave. 'M' na linha iniciada por 'I' é subsituído por 'U'. 'E' na linha iniciada por 'M' é trocado por 'Q' e assim por diante, até ter codificado toda a mensagem. No final deste procedimento, tem-se o texto

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

FIG. 3.3: Cifra de Vigenère

cifrado 'UQWBDI'. A letra 'E' aparecia 2 vezes no texto orginal. No texto codificado, não há caractere se repetindo, ou seja, a freqüência dos caracteres originais são alterados, dificultado a análise.

3.2.4.4 HOMOFÔNICOS

A substituição homofônica foi concebida para neutralizar o ataque estatístico. Essa cifra funciona buscando distribuir a aparição dos símbolos do texto claro. A substituição homofônica convencional associa um símbolo a outros, tais que a distribuição dos novos símbolos seja equiprovável. Assim, cada símbolo do alfabeto original é desdobrado em outros homofonemas, equiprováveis ou não (28).

Na substituição homofônica cada caracter da mensagem original pode ser mapeado para um ou vários caracteres na mensagem cifrada. A idéia deste tipo de substituição é precisamente evitar que se mantenha a distribuição de freqüências originais. Na distribuição homofônica de tamanho variável ou de distribuição não-equiprovável, pode-se obter um texto cifrado cuja distribuição estatística é baseado numa chave secreta, dificultando ainda mais o trabalho do criptoanalista. Nas tabelas 3.4 e 3.5 encontram-se exemplos de mapeamentos homofônicos convencionais e variáveis, respectivamente.

Caractere	Freqüência	Homofônicos
A	1/4	A'
B	3/4	B', B", B"'

TAB. 3.4: Substituição Homofônica Convencional

Caractere	Freqüência	Homofônicos
A	1/4	A'
B	3/4	B', B"

TAB. 3.5: Substituição Homofônica de Tamanho Variável

3.2.4.5 TRANSPOSIÇÃO

As cifras por transposição podem ser encontradas em jogos de quebra-cabeça de palavras, onde a mensagem original tem seus caracteres embaralhados, devendo ser remontada pemutando os quadrados com o espaço em branco. Na cifra por

transposição ou por permutação, como também é conhecida, o embaralhamento é baseado numa chave ou regra. Segue como exemplo uma cifragem baseada num retângulo para transposição.

P	O	R
D	E	U
S		M
E	N	E
Z	E	S

TAB. 3.6: Exemplo de Codificação por Transposição

Na tabela 3.6, o texto em claro 'PORDEUS MENEZES' é codificado para 'PDSEZOE NERUMES'. Para isso, basta que para isso a palavra seja lido por coluna ao invés de por linha. Para estes casos, a chave não é nada mais se não a quantidade de colunas utilizadas para montar a tabela original. Para reverter basta então escrever a palavra codificada preenchendo primeiro as colunas, depois as linhas.

Para criptoanalisar este tipo de codificação é preciso gerar todas as permutações possíveis ($N!$, onde N é o tamanho do texto) ou tentar adivinhar a chave, caso se saiba antecipadamente que o criptograma utilizado foi baseado num retângulo. O criptoanalista pode tentar alguns valores para o lado do retângulo e montar o retângulo original. Para o exemplo apresentado, seria razoável tentar chaves entre 2 e $\left\lceil \sqrt{N} \right\rceil$, neste caso, 4. Tendo sucesso na segunda tentativa ($N = 3$). Para valores elevados de N, a dificuldade para criptoanalisar as permutações cresce exponencialmente.

3.2.4.6 COMPOSIÇÃO

As cifras por composição são sistemas criptográficos compostos por mais de um método de cifragem para construir um sistema mais complexo. Esse tipo de criptografia foi criada baseada em funções matemáticas compostas. Combinando cifras de substituição com cifras de transposição pode criar um sistema mais seguro do que utilizar cada uma independentemente. Vários algoritmos modernos de criptografia utilizam este princípio. Na seção sobre criptografia moderna a seguir serão dados mais detalhes.

3.2.5 CRIPTOGRAFIA MODERNA

3.2.5.1 INTRODUÇÃO

Com o advento dos computadores, os métodos clássicos de criptografia, que já eram fáceis de serem quebrados manualmente, viraram pó ao terem seus processos de criptoanálise mecanizados, mas a necessidade de comunicar-se secretamente continuou sendo relevante, ou até teve sua importância incrementada, já que diversos meios de comunicação foram criados ou evoluídos com os computadores. Com isso, a criptografia teve que evoluir.

Assim como os métodos de criptoanálise evoluíram para um nível informatizado e passaram a quebrar em segundos os códigos criados à mão, os sistemas criptográficos também foram informatizados e passaram a combinar diversos métodos já conhecidos e também novos foram criados. Também foi possível realizar a repetição das operações quantas vezes fosse desejável, combinando a saída de uma rodada com a entrada da próxima. Essa característica permite misturar o texto a ponto de não dar qualquer indício do texto original.

A segurança deste modelo de criptografia depende de diversos fatores: força do algoritmo, força da chave, tamanho da chave, etc. O algoritmo deve ser forte o suficiente para tornar impraticável a tentativa de se descobrir seu conteúdo mesmo sem possuir a chave. A chave também deve ter sua segurança. Como dissemos, ela tem que ser forte, ou seja, não deve ser fácil de ser adivinhada. Alguns métodos de ataque baseiam-se nesses tipos de fraqueza.

3.2.5.2 SISTEMA SIMÉTRICO OU DE CHAVE SECRETA

Um sistema simétrico é assim conhecido por que a chave utilizada para encriptar a mensagem é a mesma chave utilizada em sua etapa reversa. Neste tipo de sistema criptográfico de chave única, existe a necessidade de que o remetente e o receptor entrem em acordo com relação a chave antes que possam realizar a comunicação com segurança. Portanto, a segurança do sistema criptográfico simétrico está na chave. Se a chave for de conhecimento de todos, qualquer um poderá codificar e decifrar mensagens. Assim, para que a comunicação fique secreta, a chave tem que permanecer secreta (28).

Algoritmos simétricos dividem-se em duas classes:

- **Algoritmos de fluxo:** Operam bit a bit ou byte a byte do texto claro, permitindo operações de cifragem e envio mais rápidas, contudo, oferecendo um grau menor de segurança, já que no sistema de fluxo, a operação de cifragem, normalmente, é realizado entre o bit (ou byte) do texto claro e o bit (ou byte) da chave, sem sofrer influência de outros pedaços do texto. São comumente utilizados em comunicações do tipo *streaming*, como transmissões de vídeo e áudio pela Internet;
- **Algoritmos de bloco:** Operam em um conjunto de bits (ou bytes), blocos de texto. O cálculo criptográfico é operado sobre todo o bloco de uma vez, podendo também ser mixado com outros blocos para aumentar o grau de desordem, a **entropia**, do texto criptografado. O tamanho de um bloco no algoritmo deve sempre tentar buscar o equilíbrio entre a eficiência do cálculo sobre ele com a segurança por ele oferecida, em outras palavras, ser grande o bastante para impedir uma análise e pequeno o suficiente para ser processado com eficiência (28).

No sistema de criptografia simétrica existe o problema de distribuição de chaves. Supondo que temos um número n de usuários para estabelecer conexões seguras, o número de chaves criptográficas necessárias é $n(n-1)/2$. Esta quantidade pode tornar o sistema inviável: para um sistema com 100 mil usuários, por exemplo, o número total de chaves trocadas é igual a 5 bilhões (28).

3.2.5.3 ALGORITMOS PARA CRITOGRAFIA SIMÉTRICA

Os principais algoritmos de criptografia da literatura são baseados ou na Cifra de Feistel (Rede de Feistel) (1) ou em Redes de Substituição-Permutação (SPN) (8). Na verdade a Rede de Feistel não deixa de ser uma SPN, mas com caracteristicas específicas.

A Cifra de Feistel permite que as operações de cifragem e decifragem sejam idênticas, necessitando apenas que o gerador de chaves trabalhe em modo reverso. Além desta caracteristica, também pode ser destacado a possibilidade de utilizar funções não-inversíveis (4) devido à natureza da estrutura, fortalecendo o sistema criptográfico.

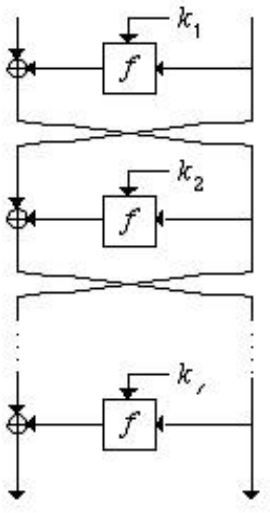


FIG. 3.4: Rede de Feistel

A figura 3.4 apresenta a estrutura da rede.

O algoritmo funciona efetuando uma determinada quantidade de rodadas (normalmente 16) sobre um bloco de dados. O bloco é dividido em duas metades (para facilitar, **R** e **L**) e cada rodada é uma operação sobre apenas um dos lados, permutando-os de lado para que na rodada seguinte o outro bloco sofra as modificações.

Como pode-se observar na figura 3.4, todas as rodadas possuem a mesma estrutura. A substituição é executada sob a metade esquerda, L, do dado de entrada. Isto é feito aplicando uma função criptográfica $\Im$ sob a metade direita, R, do dado e então efetuando uma operação de OU-Exclusivo da saída de $\Im$ e a metade L. A função $\Im$ é a mesma para todas as rodadas, mas sendo parametrizada pela subchave $\mathbf{K}_i$ da rodada. Após a etapa de substituição, a permutação consiste em trocar as duas metades R e L para a rodada seguinte. A operação pode ser representada da seguinte maneira:

$$\mathrm{L}_i = \mathrm{R}_{i-1}$$
$$\mathrm{R}_i = \mathrm{L}_{i-1} \oplus \Im(\mathrm{R}_{i-1}, \mathrm{K}_i)$$

– **DES:** O DES (*Data Encryption Satandard*) foi o primeiro padrão de criptografia adotado por um país. Em 1973 o *National Bureau of Standards*, hoje NIST (*National Institute of Standards and Technology*) lançou o convite para propostas de um padrão nacional de criptografia. O projeto da IBM foi dis-

paradamente o melhor e em 1977 o DES foi oficializado no FIPS PUB 46-2 (47) e adotado pelo governo norte-americano (4).

A estrutura (vide figura 3.5) de cifragem do DES é uma Cifra de Feistel, com o acréscimo de uma permutação inicial (PI) fixa que o texto em claro sofre antes do início das rodadas de codificação. As substituições a cada rodada dependem de matrizes fixas (*Substitution box*) e não-lienares. Ao final de 16 rodadas, as duas metades do bloco resultantes das operações sofrem mais uma nova permutação, inversa à da entrada (PI^{-1}).

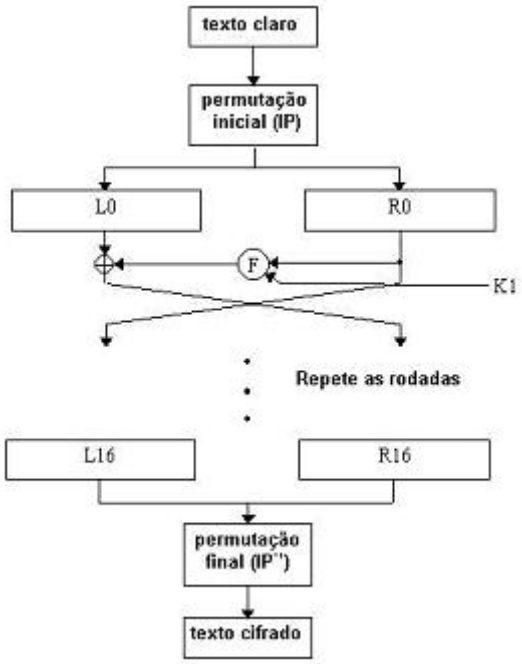


FIG. 3.5: Esquema do Data Encryption Standard

Na decifragem, assim como a de Feistel, é utilizado o mesmo esquema para cifrar, apenas revertendo a ordem das chaves geradas. Um problema da implementação desse algoritmo é que, apesar de a chave secreta possuir 64 bits, apenas 56 bits são aproveitados. Uma chave de 64 bits hoje já não garante uma segurança adequada, dependendo do tipo de informação que se deseja proteger. Pior quando, na prática, a chave é reduzida a 56 bits. Os 8 bits restantes são utilizados para detecção de erro. Detalhes em (47).

O gerador de chaves do DES recebe a chave secreta fornecida e cria 16 chaves de 48 bits, uma para cada rodada. Estas chaves são utilizadas para alimentar a função $\Im$, que opera um OU-Exclusivo sobre a metade direita do bloco depois

que este sofre uma expansão de 32 para 48 bits através de uma uma função fixa de expansão. O resultado é então tratado pelas caixas de substituição e depois misturado a partir de uma função de permutação. Os detalhes das funções de expansão, de permutação e das Caixas-S pode ser encontrado em (47) e em (4). A figura 3.6 traz o esquema simplificado da função $\Im$.

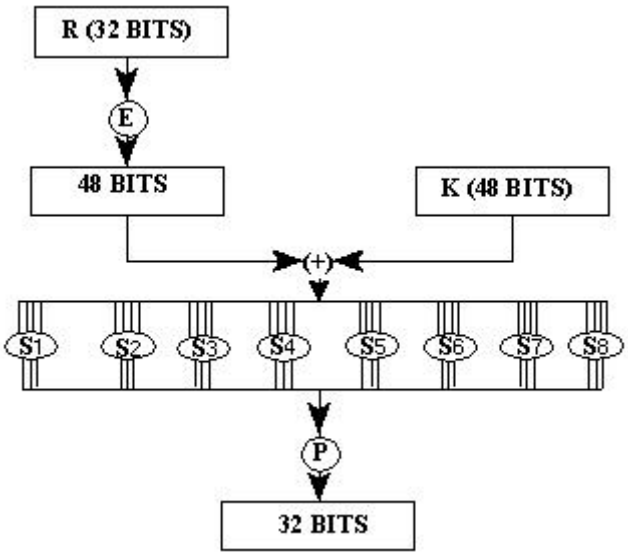


FIG. 3.6: Esquema Simplificado da Função $\Im$ do DES

– **AES:** O padrão de criptografia adotado pelo governo norte-americado em substituição ao DES é o AES (*Advanced Encryption Standard*), selecionado através de um concurso promovido pelo NIST (46). O AES (Título Original: *Rijndael*) é uma rede de substituição-permutação chamada Square (42), formada por oito rodadas, operando blocos de 128 bits e uma chave também de 128 bits (48; 43). As operações são sobre arrays 4x4 de bytes denominado *state*. Cada estágio (ou rodada) consiste em quatro passos:

  a) **AddRoundKey:** Cada byte é combinado com a chave da rodada; cada chave da rodada é derivada da chave secreta usando um gerador de chaves;

  b) **SubBytes:** Substituição não-linear onde cada byte é trocado com um outro de acordo com uma caixa de substituição fixa;

  c) **ShiftRows:** O passo de transposição ocorre com cada linha tendo seus bytes deslocados ciclicamente à esquerda (**shift left**). A quantidade de espaços destes deslocamentos diferem a cada linha;

d) **MixColumms:** Opera em colunas combinando os 4 bytes em cada coluna utilizando transformação linear através da multiplicação modular da coluna por um polinômio sob $\mathbf{GF}(2^8)$ do tipo $c(x) = 3x^3 + x^2 + x + 2$, sendo o módulo $x^4 + 1$.

A rodada final reposiciona o estágio MixColumms com outra instância do AddRoundKey.

O AES é atualmente utilizado em diversas aplicações e protocolos, como o IEEE 802.11i (WPA2), OpenSSH, WinRAR, WinZip 9, Skype, além de ser suportado por diversos outros, como o GNU Privacy Guard, NetBSD Cryptographic Disk Driver, Microsoft's .NET Framework, entre outros.

– **RC4:** O RC4 é um algoritmo de cifra de fluxo criado por Ron Rivest, que também desenvolveu diversos outros algoritmos conhecidos, como o RSA, algoritmo de chave pública. O RC4 (*Rivest Cipher 4*) é o algoritmo de fluxo utilizado no SSL (*Secure Socket Layer*), protocolo que atua em baixo da camada de aplicação http, formando o https. Outro protocolo que utiliza o RC4 é o WEP (*Wired Equivalent Privacy*), criado para prover segurança em redes sem-fio IEEE 802.11b (38). O RC4 porém caiu em descrédito depois de diversas publicações (39; 40; 41) sobre suas deficiências.

  O algoritmo trabalha gerando uma cadeia de bits pseudoaleatórios que é combinada com o texto claro utilizando XOR. A geração da cadeia de chave contínua depende da permutação de todas os 256 bytes possíveis e de dois ponteiros de 8 bits. A estrutura interna do gerador utiliza um sistema gerador de chaves (KSA: *Key-Scheduling Algorithm*) e de um gerador de números pseudoaleatórios (PRGA: *pseudo-random generation algorithm*). A decifragem opera da mesma forma.

– **RC5:** O RC5 (*Rivest Cipher 5*) é a quinta versão de uma seqüência de melhoramentos em algoritmos anteriores de Ron Rivest. Ele segue uma estrutura modificada de Feistel e foi concebido para possuir determinadas características (4): desempenho, aplicabilidade tanto em hardware como em software, adaptável a processadores com diferentes comprimentos de palavras, número de rodadas ajustável, chave de tamanho variável, de implementação simples, baixa utilização de memória, alta segurança e rotações dependentes dos dados. Os

parâmetros a serem ajustados no RC5 são o tamanho do bloco a ser utilizado, o número de rodadas e o tamanho da chave secreta.

A cifragem do RC5 é baseada em operações primitivas: adição e módulo, OU-Exclusivo e rotação circular a esquerda. A simplicidade do algoritmo é tamanha que o pseudocódigo abaixo representa toda a cifragem:

$LE_0 = A + S[0]$
$RE_0 = B + S[1]$
para i = 1..k

$$LE_i = ((LE_{i-1} \oplus RE_{i-1}) \prec\prec\prec RE_{i-1}) + S[2_i]$$
$$RE_i = ((RE_{i-1} \oplus LE_{i-1}) \prec\prec\prec LE_i) + S[2_i + 1]$$

A decifragem não utiliza o mesmo algoritmo, mas é facilmente derivável do de cifragem, executando as operações em ordem inversa.

3.2.6 TIPOS DE ATAQUE

O ataque por Força Bruta é o mais comum e é normalmente utilizado como um parâmetro de segurança do sistema criptográfico. Este ataque dá-se executando exaustivamente todas as possibilidades se decodificar o texto criptografado, ou tentando adivinhar a chave ou o próprio texto em claro original.

Na tentativa de quebrar a chave, o atacante deve ter em mãos o algoritmo utilizado e o texto cifrado e gerar todas as decodificações com todas as chaves possíveis até que um dos textos faça sentido e assim tenha sido gerado o texto claro correto.

Uma outra possibilidade é tentar adivinhar o texto analisando os passos do algoritmo utilizado. Isso se daria estudando quais possíveis codificações um caractere a ser criptografado pode gerar durante o processo de cifragem. É desejável que o esforço computacional necessário para executar uma quebra por força bruta baseado nesta abordagem seja mais elevado que o esforço para a quebra da chave, já que é nela que deve estar a segurança do sistema criptográfico.

Serão dados mais detalhes sobre este método de quebra sobre o texto no capítulo sobre a segurança do sistema a ser proposto aqui.

A seguir as modalidades de um ataque por força bruta. Os textos foram retirados de (28):

a) **Ataque por texto cifrado:** O criptoanalista tem acesso a grande quantidade de mensagens cifradas com um determinado algoritmo, mas desconhece as mensagens originais e as chaves utilizadas. Sua tarefa é recuperar as mensagens originais ou, melhor ainda, deduzir as chaves utilizadas.

   Neste tipo de ataque, uma técnica que pode ser utilizada com sucesso é o chamado ataque estatístico. É baseada na constatação que em vários textos de um mesmo idioma a freqüência de cada letra é mais ou menos fixa. O criptoanalista usa este conhecimento para comparar a freqüência de cada letra da mensagem cifrada com as freqüências no idioma suposto. Se a codificação foi feita através de uma simples substituição, a quebra é imediata.

b) **Ataque por texto original conhecido:** O criptoanalista não tem somente acesso a uma grande quantidade de mensagens cifradas, mas conhece também o texto original dessas mensagens. O trabalho do criptoanalista é deduzir as chaves utilizadas ou um algoritmo para decifrar as mensagens com a mesma chave.

c) **Ataque por texto original escolhido:** O criptoanalista não só tem acesso as mensagens cifradas com as respectivas mensagens originais, como também pode escolher uma determinada mensagem e obter o texto cifrado. O trabalho do criptoanalista é deduzir as chaves utilizadas ou o algoritmo para decifrar novas mensagens cifradas com a mesma chave.

d) **Ataque adaptativo por texto original escolhido:** Este é um caso especial do anterior. Neste ataque o criptoanalista não só pode escolher as mensagens que serão cifradas mas também pode estudar os resultados e submeter novas mensagens, ou seja, permite a realimentação.

e) **Ataque por texto cifrado escolhido:** O criptoanalista pode escolher diferentes mensagens cifradas e obter as mensagens originais correspondentes. Este ataque é utilizado quando se tem uma máquina que faz decifragem automática. O trabalho do criptoanalista é deduzir a chave.

3.2.6.1 CRIPTOANÁLISE LINEAR

Criptoanálise linear foi desenvolvida por Mitsuru Matsui (12; 13) e é um tipo de ataque que usa aproximações lineares para descrever a ação do DES. Aproximações

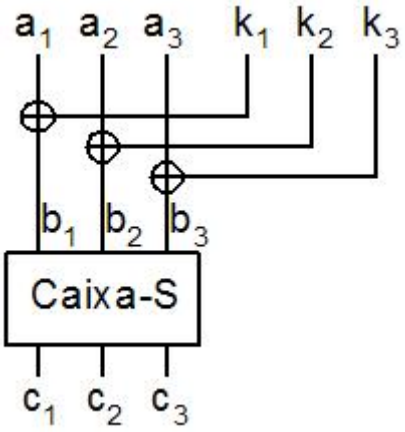


FIG. 3.7: Exemplo de Caixa de Substituição

Entrada B (b_1, b_2, b_3)	Saída C (c_1, c_2, c_3)
000	001
001	100
010	010
011	100
100	101
101	011
110	101
111	000

TAB. 3.7: Saída da Caixa-S

lineares, ou correlações lineares, são equações lineares que valem com alguma probabilidade p. Este tópico é baseado em (14).

Seja a figura 3.7 a representação de uma cifra de substituição, chamada de Caixa-S (*S-Box*). Nesta cifra, é feita uma operação XOR entre os bits do texto em claro A (a_1, a_2, a_3) e os bits da chave secreta K (k_1, k_2, k_3) e o resultado é transformado pela *S-Box* gerando os bits de saída C (c_1, c_2, c_3) conforme a tabela 3.7.

Então, $C = S(B) = S(A \oplus K)$. Da tabela, podem ser obtidas as seguintes relações (ou correlações):

- $b_1 \oplus b_3 = c_1$, sempre, i.e., com probabilidade $p = 1$. Como $b_1 = a_1 \oplus k_1$ e $b_3 = a_3 \oplus k_3$, logo $a_1 \oplus k_1 \oplus a_3 \oplus k_3 = c_1$, ou $k_1 \oplus k_3 = c_1 \oplus a_1 \oplus a_3$. Isso equivale a dizer que, caso o texto em claro e o respectivo cifrado sejam conhecidos, pode-se calcular o valor da expressão $k1 \oplus k3$. Então, se $k1 \oplus k3 = 1$, por exemplo, tem-se 01 ou 10 para o par, e caso $k1 \oplus k3 = 0$, tem-se 00 ou 11.

Ou seja, a busca é reduzida pela metade, é como se um bit da chave fosse conhecido.

- $b_1 \oplus b_2 = c_2$, vale para 6 das 8 entradas possíveis, i.e., possui uma probabilidade $p = 3/4$, caso as entradas ai sejam independentes e aleatórias. Logo, a equação $k_1 \oplus k_2 = c_2 \oplus a_1 \oplus a_2$ acontecerá 75% das vezes, e basta conhecer alguns pares de textos em claro e os respectivos textos cifrados que o resultado correto de $k_1 \oplus k_2$ ocorrerá com essa probabilidade p.
- $b_1 = c_3$, também vale para 6 das 8 entradas possíveis. Logo, $k_1 = c_3 \oplus a_1$ também ocorre com $p = 3/4$.

Daí tem-se as equações:

a) $k_1 \oplus k_3 = c_1 \oplus a_1 \oplus a_3$, $p = 1$;

b) $k_1 \oplus k_2 = c_2 \oplus a_1 \oplus a_2$, $p = 3/4$;

c) $k_1 = c_3 \oplus a_1$, $p = 3/4$.

De posse dos valores obtidos empiricamente através das probabilidades associadas consegue-se o valor da chave. Na verdade, uma equação linear obtida do algoritmo só será útil se $p \neq \frac{1}{2}$, e será o mais eficiente quanto maior o valor da expressão $\left|p - \frac{1}{2}\right|$.

3.2.6.2 CRIPTOANÁLISE DIFERENCIAL

A criptoanálise diferencial é um método que consiste em se analisar o efeito que diferenças entre dois textos em claro irão causar nas diferenças entre os dois textos cifrados resultantes. Estas diferenças são usadas para descobrir conjuntos de chaves possíveis. Este tópico também é baseado em (14).

Considere novamente a figura 3.7 e a tabela 3.7 como representação de uma Caixa de Substituição.

Notação:

- A (a_1, a_2, a_3) = texto em claro
- A*(a_1*, a_2*, a_3*) = segundo texto em claro
- K (k_1, k_2, k_3) = chave secreta

– B (b_1, b_2, b_3) = entrada na Caixa-S

– B*(b_1*, b_2*, b_3*) = segunda entrada na Caixa-S

– $B' = B \oplus B^*$ = diferença entre as entradas

– C $(c_1, c_2, c_3) = S(B) = S(A \oplus K)$, saída da Caixa-S

– C*$(c_1*, c_2*, c_3*) = S(B*) = S(A* \oplus K)$, segunda saída da Caixa-S

– C'= $C \oplus C^*$ = diferença entre as saídas

Então, para cada par de entradas B, B* tem-se o valor da diferença B'. Analogamente, tem-se C' como a diferença entre as respectivas saídas C, C*, com $C = S(B) = S(A \oplus K)$ e $C* = S(B*) = S(A* \oplus K)$. A tabela 3.8 mostra o número de ocorrências de valores C' para cada valor possível de B'.

	$C' = C \oplus C*$							
$B' = B \oplus B*$	000	001	010	011	100	101	110	111
000	8	0	0	0	0	0	0	0
001	0	0	0	0	0	4	4	0
010	4	0	0	4	0	0	0	0
011	0	0	0	0	0	4	4	0
100	0	0	0	0	4	0	0	4
101	0	4	4	0	0	0	0	0
110	0	0	0	0	4	0	0	4
111	0	8	8	0	0	0	0	0

TAB. 3.8: Número de ocorrências do valor de saída C' para cada entrada B'

Diz-se que $X \Rightarrow Y$, ou X implica em Y pela Caixa de Substituição, se o elemento da tabela (X, Y) é diferente de 0, i.e., existe pelo menos um par B,B* que possui diferença B'=X, tais que as saídas C,C* possuam diferença C'=Y.

Da tabela reftab:diferencial, tem-se as implicações possíveis:

– 000 $\Rightarrow$ 000;

– 001 $\Rightarrow$ 101 ou 001 $\Rightarrow$ 110;

– 010 $\Rightarrow$ 000 ou 010 $\Rightarrow$ 011;

– 011 $\Rightarrow$ 101 ou 011 $\Rightarrow$ 110;

– 100 $\Rightarrow$ 100 ou 100 $\Rightarrow$ 111;

– 101 $\Rightarrow$ 001 ou 101 $\Rightarrow$ 010;

- $110 \Rightarrow 100$ ou $110 \Rightarrow 111$;
- $111 \Rightarrow 001$.

Para se descobrir o valor da chave K, o procedimento é o seguinte. Suponha dois pares de textos A, A*. Logo, as entradas são $B = A \oplus K$ e $B* = A* \oplus K$. E, tem-se que $B' = B \oplus B* = A \oplus K \oplus A* \oplus K = A \oplus A*$. Então, uma vez que o par A, A* for escolhido, a Caixa de Substituição dará como resposta, em função da chave K, pares de resposta C, C*. Então, dependendo de $C' = C \oplus C*$ tem-se os pares possíveis para B, B* e o conjunto de chaves prováveis pelas equações: $K = A \oplus B = A* \oplus B*$.

Exemplo: seja K=100 a chave secreta escolhida que se quer descobrir. Então, escolhe-se, por exemplo, os pares de texto em claro A=011 e A*=010, então $B' = A \oplus A* = 001$. Mas se B'=001 então tem-se apenas dois valores possíveis para C': $001 \Rightarrow 101$ ou $001 \Rightarrow 110$. Suponha que a Caixa-S forneceu como saída valores C, C* que resultaram em C'=101 por exemplo. Então, tem-se apenas 4 valores possíveis de pares B, B* possíveis. Note que os pares B,B* são duais, i.e., se existe B1, B2, então também existe o par B2, B1. Os pares são os seguintes: 000-001, 111-110 e os inversos. Logo, tem-se 4 valores possíveis para a chave $K = A \oplus B = A* \oplus B*$:

- $K1 = 011 \oplus 000 = 010 \oplus 001 = 011$
- $K2 = 011 \oplus 111 = 010 \oplus 110 = 100$
- $K3 = 011 \oplus 001 = 010 \oplus 000 = 010$
- $K4 = 011 \oplus 110 = 010 \oplus 111 = 101$

Repetindo esse procedimento para A=100 e A*=000, tem-se $B' = A \oplus A* = 100$. Logo as implicações possíveis são $100 \Rightarrow 100$ ou $100 \Rightarrow 111$. Suponha que a Caixa de Substituição resultou em C'=100. Então, teremos da mesma forma 4 pares possíveis (na verdade, são 2 pares e seus duais) e 4 chaves possíveis também:

- $K1 = 100 \oplus 000 = 000 \oplus 100 = 100$
- $K2 = 100 \oplus 111 = 000 \oplus 011 = 011$
- $K3 = 100 \oplus 100 = 000 \oplus 000 = 000$
- $K4 = 100 \oplus 011 = 000 \oplus 111 = 111$

Dos 2 experimentos, tem-se que a única chave provável que aparece nos dois conjuntos de solução é K=100. Caso houvesse mais de uma chave coincidente, o processo deve ser repetido o número de vezes que for necessário até se encontrar apenas 1 chave em comum entre todos os conjuntos de solução.

4 SISTEMAS COMPRESSORES

4.1 INTRODUÇÃO

Um texto em formato digital, ou seja, salvo em arquivo de computador, como um 'doc' ou um 'txt', é formado por dezenas, centenas e até milhares de bytes, cada um representando um caractere desse texto. Suponha que se deseje armazenar um arquivo de grande porte em algum tipo de memória, primária ou secundária. Para melhor utilizar os recursos disponíveis, deseja-se também minimizar, de alguma forma e na medida do possível, o espaço de memória utilizado. Uma forma de tentar resolver esse problema consiste em codificar o conteúdo do arquivo de maneira apropriada. Se o arquivo codificado for menor que o original, pode-se armazenar a versão codificada em vez do arquivo propriamente dito. Isto representaria uma economia de memória. Naturalmente, uma tabela de códigos seria também armazenada, para permitir a decodificação do arquivo. Essa tabela seria utilizada pelos *algoritmos de codificação* e *decodificação*, os quais cumpririam a tarefa de realizar tais operações de forma automática. O problema descrito é conhecido como *compressão de dados* (11).

A compressão de dados objetiva reduzir o espaço ocupado por este arquivo, aproveitando as redundâncias (repetições) do texto, buscando indexá-las de modo que quando ocorra uma repetição, essa possa ser substituída por uma codificação de tamanho menor, mas sendo possível uma reversão de maneira que a informação contida do documento possa ser obtida, executando-se a operação de reversão.

Diversos algoritmos de compressão foram desenvolvidos com diferentes abordagens para se aproveitar das repetições textuais, de modo a reduzir a quantidade de bits necessários para representar um byte. Existem diversos tipos de algoritmos, alguns são específicos para determinados tipos de arquivos, como imagens e vídeos, mas este trabalho foca apenas nos algoritmos para compressão de dados textuais.

São várias as possibilidades de tratar as redundâncias de um texto. Pode-se classificar estas redundâncias por repetições de letras, de conjunto de duas ou três letras

ou, palavras completas ou mesmo blocos de texto. A idéia é sempre eliminar o máximo possível das redundâncias, reduzindo o tamanho final da informação. Um exemplo de eliminação fácil de compreender é o da repetição de letras. A seqüência 'AAABBBCDDDDD' ocupa 12 bytes de dados, mas é visível as repetições contidas: repetições das letras 'A','B' e 'D'. Uma maneira de eliminar essa redundância seria substiuir as repetições com a informação de quantas vezes cada letra está se repetindo em seqüência. Então, para o exexemplo dado, tem-se como resultado a seqüência '3A3BC5D', que ocupa 7 bytes.

A importância da compressão de dados era mais considerável até o final da década de 1980, quando o preço do *megabyte* era bastante elevado. Naquela época, a questão de economia de memória era crítica. Com o passar do tempo, a memória foi se tornando cada vez mais abundante, com o relativo decréscimo do custo. O custo de um *megabyte* em 1956 chegava a US$10.000. A figura 4.1 apresenta a redução do preço por *megabyte* desde o início da década de 1980. Esse barateamento deve-se principalmente à evolução da tecnologia, que permitiu que se armazene cada vez mais informações em discos de tamanho físico menor. A figura 4.2 traz a evolução da capacidade de armazenamento do disco rígido com o passar dos anos.

Entretanto, alguns programas de aplicação atualmente utilizados requerem muita memória. Entre esses, por exemplo, encontram-se programas que oferecem recursos visuais aos seus usuários. Assim, o problema de economia de memória permanece atual (11).

Com o barateamento da infraestrutura necessária para guardar informação, algoritmos de compressão de dados deveriam ter perdido um pouco a importância, mas não foi o que aconteceu. O advento e a larga utilização das redes de computadores também contríbuem para a importância da compressão de dados. Agora, além da economia de memória, procura-se também diminuir o custo de transmissão de arquivos na rede. Ou seja, uma maior compressão de dados de um arquivo corresponderia a um número menor de dados a transmitir, o que implicaria um custo de transmissão menor. A hipótese concreta, nesse caso, é que o custo de transmissão é proporcional à quantidade de dados a transmitir (11).

Os algoritmos de compressão podem ser classificados de uma maneira geral como '**com perda de informação**' e '**sem perda de informação**' (16). O método

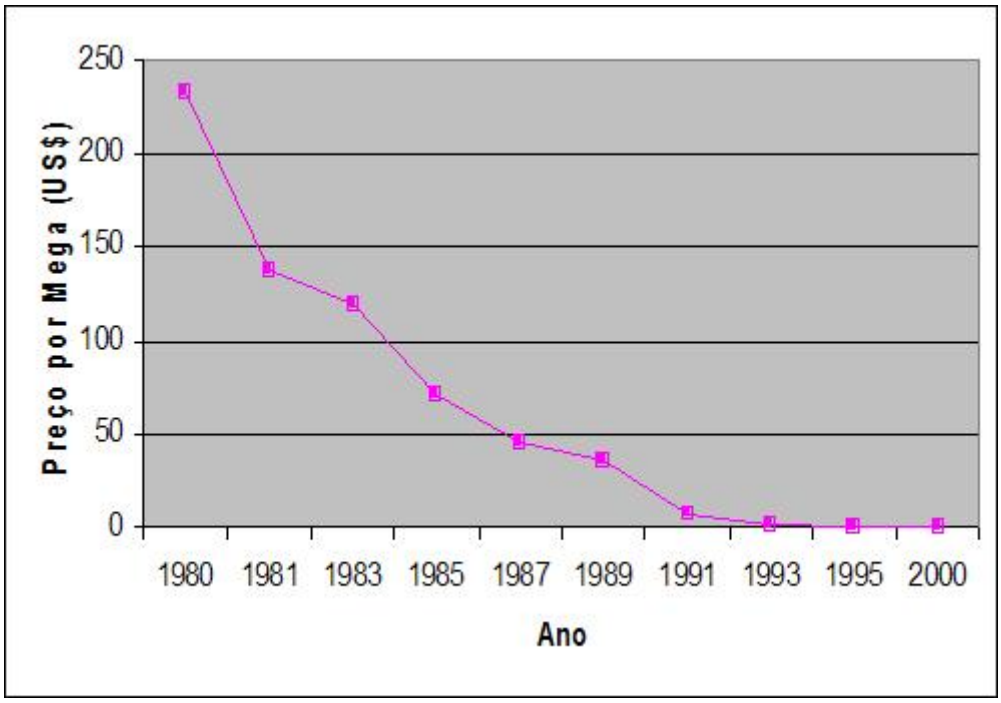


FIG. 4.1: Evolução do Preço por MB

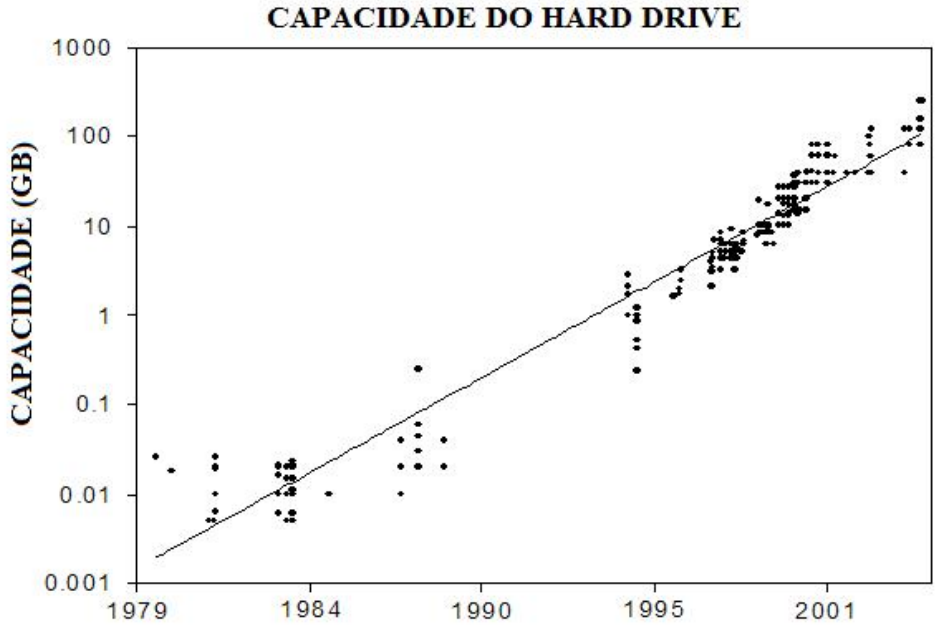


FIG. 4.2: Evolução da Capacidade de Armazenamento

é sem perdas quando a informação codificada é recuperada integralmente após a decodificação. Este tipo de aplicação é importante quando perdas de informação podem acarretar no não funcionamento de um código executável ou na perda de informações financeiras, que são dados que precisam ser totalmente íntegros.

Há situações onde nem toda informação é totalmente necessária. Isso ocorre quando um dado analógico é convertido para um formato digital. Por exemplo, na codificação de um áudio, pode aparecer freqüências muito altas ou muito baixas para que seja perceptível ao ser humano, e podem ser descartadas. Os métodos que codificam a informação desta maneira são conhecidos como algoritmos com perda de informação.

Outras classificações que podem ser empregadas são (16):

- **Simétricos:** Quando o algoritmo de compressão é o mesmo de descompressão e a complexidade de ambos é igual, ou seja, se o método utilizado para descomprimir é o mesmo ou pelo menos possui poucas modificações em relação ao método para comprimir, estes são chamados *simétricos*. São exemplos de algoritmos simétricos o LZW e a codificação aritmética (18). Se os algoritmos de compressão e descompressão são bastante diferentes e com complexidades diferentes, estes são conhecidos por *assimétricos*. Um exemplo de assimétrico é o LZ77 (6).
- **Adaptabilidade:** O compressor é *adaptativo* quando seu método varia de acordo com a leitura dos dados, ou *não-adaptativo* quando o método é fixo, não importando os dados lidos. São exemplos de códigos adaptativos o LZ77 e o LZ78 (6).
- **Fluxo ou Bloco:** Classifica-se um compressor do tipo de bloco quando o método atua sobre um agrupamento (bloco) de dados para melhorar a compressão. O método de Burrows-Wheeler, foco principal deste trabalho, é um exemplo. A maioria dos algoritmos tradicionais são de fluxo.
- **Operacionabilidade:**

  Estatísticos: São baseados na probabilidade da ocorrência dos símbolos ou na contagem direta dos símbolos.

  de Dicionário: São os que utilizam estruturas de dados, como dicionários, para auxiliar na eliminação das redundâncias.

Transformações: Não comprimem os dados diretamente, mas auxiliam na melhoria dos resultados de outros métodos.

4.2 EXEMPLOS DE SISTEMAS COMPRESSORES

4.2.1 CODIFICAÇÃO *RUN-LENGTH*

O *Run-Length Encoding*(16), ou RLE, é uma técnica de compressão simples e intuitiva. O RLE trata de substituir as repetições consecutivas de uma seqüência de caracteres pelo número que informa quantidade de repetições seguido (ou até precedido, dependendo da implementação) do caractere repetido.

No exemplo apresentado no início deste capítulo, a expressão 'AAABBBCDDDDD' é codificada para '3A3BC5D', numa demonstração típica do funcionamento do RLE. O problema nesta codificação é distinguir o que é número no texto e o que é número indicando as repetições. Para resolver isso é necessário utilizar um caractere especial para indicar o início do que é codificado. Outra abordagem utilizada é a alocação do dígito após a seqüência, algo como 'aaa1', indicando uma seqüência de 4 caracteres 'a'. A implementação mais atual para o RLE é a substituição das repetições por tuplas que indicam a quantidade de repetições e o caractere repetido. Também é válido a substituição de bigramas ou trigramas com o RLE.

$$\text{'AAABBB555C333DDDDD22'} \overset{RLE}{\rightarrow} (3,'A')(3,'B')(3,'5')C(3,'3')(5,'D')22$$

O problema da estratégia deste compressor é que em uma linguagem natural não é comum que ocorram mais do que duas repetições de um mesmo caractere em seqüência, por isso, raras as vezes este compressor é utilizado isoladamente. Algum tipo de permutação reversível é necessária para agrupar valores iguais e assim incrementar o resultado da compressão final. Um método que pode ser aplicado é a Transformada de Burrows-Wheeler, que será apresentada no capítulo 5.

4.2.2 CÓDIGO DE HUFFMAN

Uma das melhores técnicas de compressão conhecidas é devida a Huffman. Para uma dada distribuição de probabilidades gera um código ótimo, livre de prefixo e

Símbolo	freqüências
0	0,20
1	0,25
2	0,15
3	0,08
4	0,07
5	0,06
6	0,05
7	0,05
8	0,05
9	0,04

TAB. 4.1: Tabela de freqüências por Símbolo

de redundância mínima, além de produzir uma seqüência de bits aleatórios. Utiliza códigos de tamanho variável para representar os símbolos do texto, que podem ser caracteres ou cadeias de caracteres (digramas, trigramas, n-gramas ou palavras). A idéia básica do algoritmo é atribuir códigos de bits menores para os símbolos mais frequentes no texto e códigos mais longos para os mais raros (35).

Em suma, o código de Huffman trabalha contando a freqüência dos caracteres no texto a ser codificado e os ordena de acordo com a contagem. Os caracteres que mais aparecem no texto recebem uma representação binária, um código, menor do que a dos demais. Uma árvore (a Árvore de Huffman) é criada a partir deste *ranking*, cada caractere é subsituído por essa representação.

Tome como exemplo um texto que possua somente algarismos, cuja freqüência obedeça a tabela 4.1. Esta tabela gera a árvore apresentada em 4.3.

A geração da árvore é a primeira fase do Código de Huffman, a segunda é a execução da codificação pela substituição da simbologia binária criada. Para achar o código de cada símbolo, basta navegar na árvore de Huffman da raiz até a folha, concatenando os valores encontrados para cada nó da árvore: bit 0 (zero) ao navegar na sub-árvore da esquerda, e bit 1 ao navegar na sub-árvore da direita. A tabela 4.2 lista os códigos de Huffman resultantes. A partir do resultado, sendo '2008 ', de 4 bytes (ou 32 bits), um possível trecho do texto, ele seria codificado como 11000000101, totalizando 11 bits.

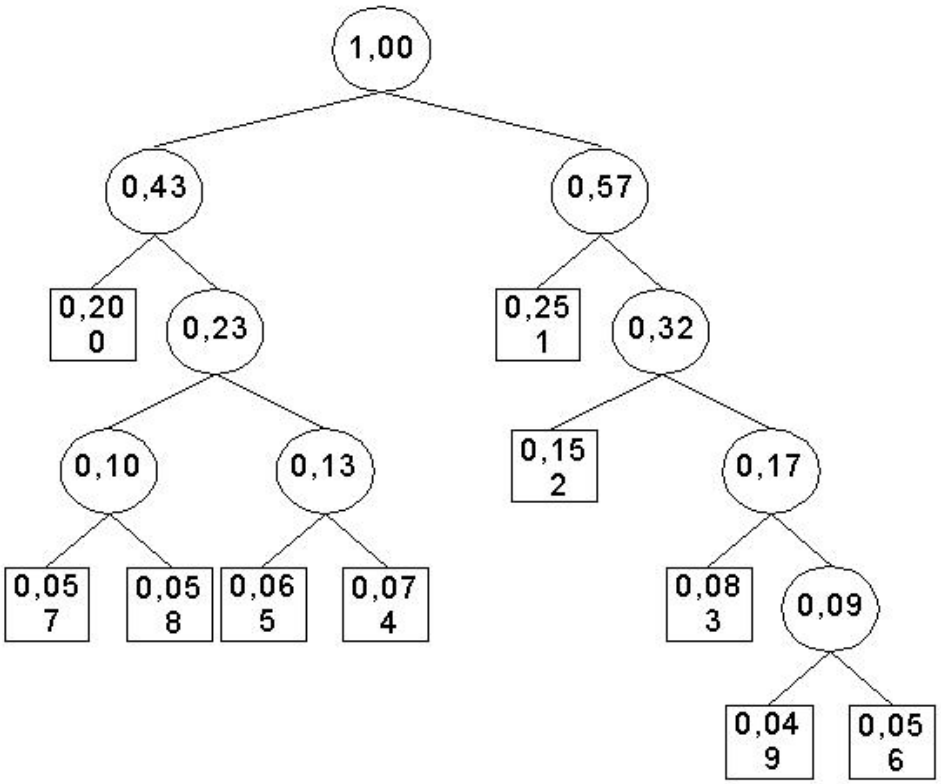


FIG. 4.3: Árvore de Huffman para a Tabela 4.1

Símbolo	freqüências
0	00
1	01
2	110
3	1110
4	0111
5	0110
6	11111
7	0100
8	0101
9	11110

TAB. 4.2: Tabela de Códigos de Huffman por Símbolo

⇓ Aponta													
c	a	b	r	a	c	a	d	a	b	r	a	r	r
Janela de Busca								Janela Adiante					

TAB. 4.3: Funcionamento do LZ77

4.2.3 LZ77

O LZ77 (17) é um algoritmo eficiente para compressão de dados e útil em redes de computadores para economizar espaço e tempo de transmissão, assim também como economia de espaço em mídia eletrônica. Variações deste algoritmo são implementadas e conhecidas como PKzip, Zip, LHarc e ARJ (6).

O LZ77 é um algoritmo fácil de implementar e a decodificação é realizada de maneira rápida e requer pouca memória, além de poder ser configurado para utilizar menos ou mais recursos, dependendo do que estiver disponível e tornando-se viável para aplicação em dispositivos de baixa capacidade de processamento.

O compressor baseia-se na reutilização de partes previamente lidas do texto, substituindo as ocorrências seguintes pelas coincidentes de seqüências anteriores. O espaço de busca e de endereçamento são limitados por uma 'janela deslizante' (*sliding window*) de tamanho fixo, mas parametrizável, que navega sobre o texto, delimitando o início e o fim da área de busca para padrões coincidentes. Obviamente, quanto maior a seqüência substituída, maior a taxa de compressão resultante.

A saída consiste em um conjunto de triplas, cuja posições indicam onde e como substituir os textos, além de trazer a primeira letra da seqüência seguinte. A janela de atuação desloca-se pelo texto 'apontando' para as seqüências anteriores. Para melhor entendimento da execução, será ilustrado o exemplo apresentado em (6). Para uma seqüência S='abracadabra', quando 'abra' aparecer pela segunda vez, ela será substituída por uma informação (tupla) que indica sua ocorrência prévia, substituindo essa expressão por um valor de menor tamanho.

A seqüência em 4.3 é analisada pelo LZ77 através de duas janelas deslizantes: uma para ler a seqüência já codificada e marcar onde inicia a coincidência dos caracteres, conhecida por **Janela de Busca**, e a outra para a seqüência a ser substituída, **Janela Adiante**. As janelas possuem tamanho k e l, respectivamente, com $k \geq l$ e fazem parte da configuração do sistema compressor. Para o exemplo da tabela 4.3,

a Janela de Busca tem comprimento 7 e a Adiante, 6. Para implementações reais, essas janelas possuem valores maiores.

A informação codificada é transformada numa tupla do tipo $< d, c, C(S) >$, onde d é o valor do deslocamento entre o fim da Janela de Busca e o apontador, que indica onde a seqüência coincidente se encontra, c é a quantidade de caracteres coincidentes e $C(S)$ é o primeiro caractere pós-seqüência na Janela Adiante.

No exemplo dado em 4.3, o apontador indica a primeira letra dentro da Janela de Busca que é igual ao primeiro caractere da Janela Adiante, no caso o 'a'. Vê-se que além do 'a', os 3 próximos caracteres da Busca coincídem com os também 3 seguintes ao 'a' na Adiante. d então é calculado pela posição do 'a' dentro da Janela até o primeiro caractere da Janela Adiante, $d = 7$. $c = 4$, representando o tamanho da seqüência 'abra', e $C(S) = C('r')$ já que 'r' é o primeiro caractere após esta seqüência na Janela Adiante.

Na verdade, o valor de d é calculado do final para o começo da Janela de Busca, objetivando sempre obter o maior c. Para o exemplo apresentado, tem-se que o último 'a' da Busca, $d = 2$, resulta em $c = 1$. O penúltimo 'a', com $d = 4$, resulta em $c = 1$ também. Analisando mais a esquerda tem-se um 'a' de deslocamento $d = 7$ gerando um $c = 4$. Com isso, a saída do LZ77 para este caso é $< 7, 4,' r' >$. Após essa codificação, as janela são deslocadas em $c + 1$ posições.

Para explicar o funcionamento da descompressão do LZ77, suponha que já se tenha decodificado os códigos correspondentes a 'cabraca', e o LZ77 tem a seguir os códigos:

$$< 0, 0,' d' >, < 7, 4,' r' >$$

A primeira tupla é facilmente descomprimida para 'd', obtendo 'cabracad', já que o primeiro zero a esquerda indica que não há deslocamento e com isso, não há texto repetido. A seguir tem-se a tupla $< 7, 4,' r' >$, indicando que o cursor deve retornar 7 posições e copiar 4 caracteres a partir da posição obtida. Ao final da seqüência de passos apresentada através das tabelas 4.4, 4.5, 4.6, 4.7 e 4.8, basta anexar o 'r' ao final da saída, obtendo 'cabracadabrar'.

⇓ Aponta 7 esq.							
c	a	b	r	a	c	a	d

TAB. 4.4: Descompressão do LZ77: Apontador volta 7 espaços

⇓							⇓
c	a	b	r	a	c	a	d	a

TAB. 4.5: Descompressão do LZ77: Copia 1

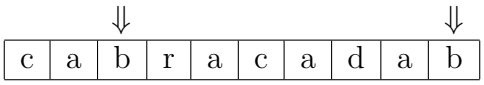
4.2.4 CODIFICAÇÃO ARITMÉTICA

Codificação Aritmética é uma técnica que torna possível obter excelentes compressões utilizando modelos sofisticados. A principal força é poder codificar arbritariamente próximo ao valor de entropia. De acordo com Shannon em (7) não é possível uma codificação melhor que o valor de entropia, então, por este aspecto, a codificação aritmética é ótima (18).

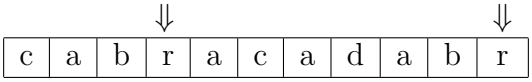
A codificação é baseada na estatística da utilização dos caracteres, mas não utiliza associação entre caracteres e códigos, assim com o de Huffman, e consiste em codificar os símbolos utilizando o intervalo $[0, 1)$. Inicialmente, a probabilidade de ocorrência de todo e qualquer símbolo é assumida ser a mesma. A cada símbolo lido, os valores de probabilidade são corrigidos e o intervalo onde se encontra o símbolo lido é subdividido. Esse passo é repetido até não haver o que ser lido. A saída é um valor qualquer entre os valores que representam o intervalo resultante de todas as subdivisões geradas pela leitura dos símbolos.

Como exemplo para facilitar o entendimento sobre a codificação aritmética, seja $s =' bcca'$ a seqüência de caracteres que se deseja comprimir e $\Sigma = (a, b, c)$ o alfabeto ternário disponível. Logo, inicialmente, os caracteres terão distribuição de probabilidade iguais a 1/3. Após a leitura de 'b', o intervalo $[0.3333, 0.6667)$ que o representa é selecionado, substituindo o $[0, 1)$ original. Vide figura 4.4.

Antes da codificação do primeiro 'c', a distribuição de probabilidades é adaptada à nova realidade, pois 'b' já foi visto. Nesse caso, os novos valores são $p(a) = 1/4$, $p(b) = 2/4$ e $p(c) = 1/4$, onde $p(x)$ é a probabilidade de ocorrência de x. O subintervalo $[0.3333, 0.6667)$ é particionado em $[0.3333, 0.4167)$, $[0.4167, 0.5834)$ e $[0.5834, 0.6667)$, representando 'a', 'b' e 'c' respectivamente. O novo subintervalo selecionado é $[0.5834, 0.6667)$, por representar o símbolo 'c' lido, figura 4.5. O mesmo



TAB. 4.6: Descompressão do LZ77: Copia 2



TAB. 4.7: Descompressão do LZ77: Copia 3

procedimento é executado para as próximas codificações, até que ao final reste o intervalo $[0.6334, 0.6390)$ que representa o intervalo para 'a' depois de sua leitura. A saída da compressão é qualquer valor pertencente a este intervalo, como a média aritmética dos extremos (0.6362 neste caso).

Para efetuar a descompressão, o decodificador simula o que o codificador deve ter feito, iniciando com o intervalo original $[0, 1)$ e dividindo os intervalos da mesma maneira que na codificação mostrada nas figuras 4.4, 4.5, 4.6 e 4.7.

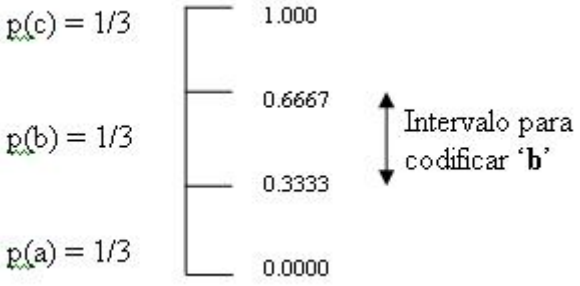


FIG. 4.4: Codificação de '**b**'

				⇓							⇓
c	a	b	r	a	c	a	d	a	b	r	a

TAB. 4.8: Descompressão do LZ77: Copia 4

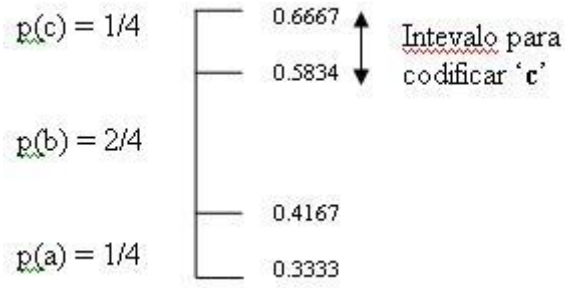


FIG. 4.5: Codificação do Primeiro '**c**'

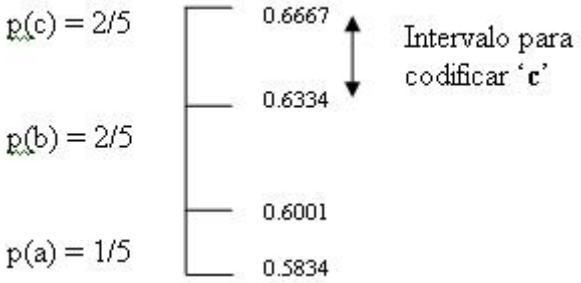


FIG. 4.6: Codificação do Segundo '**c**'

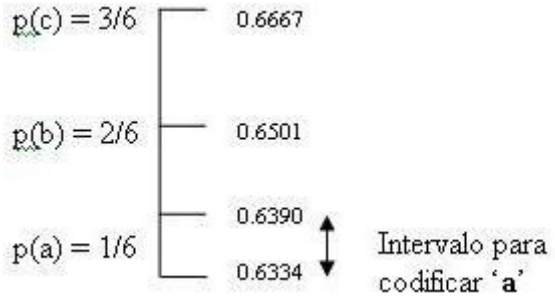


FIG. 4.7: Codificação de'**a**'

5 ALGORITMO DE COMPRESSÃO BLOCK-SORTING

O algoritmo de compressão *Block-Sorting*, também conhecido como Algoritmo de Compressão pela Transformada de Burrows-Wheeler (*Burrows-Wheeler Transform Compression Algorithm* - BWTCA), trabalha sobre o texto a ser comprimido de modo a alterar as propriedades estatísticas deste, visando incrementar as repetições de valores (freqüências dos caracteres). Essas caracteristícas influenciam compressores probabilisticos, tais como Huffman e *Run-Length Encoding*. Ou seja, isoladamente, o *Block-Sorting* não é um compressor de dados, e sim apenas um formatador de entrada para tornar o texto *mais adequado* para a compressão.

Implementações do *Block-Sorting Compression Algorithm* são encontradas no BZip, BZip2 e no Winzip versão 11.

O *Block-Sorting* é dividido em três etapas, mostradas na figura 5.1. A primeira é a transformação do texto através de permutações, para que os caracteres iguais fiquem localizados o mais contiguamente possível, preferencialmente em seqüência. Esta primeira etapa é conhecida por **Transformada de Burrows-Wheeler**.

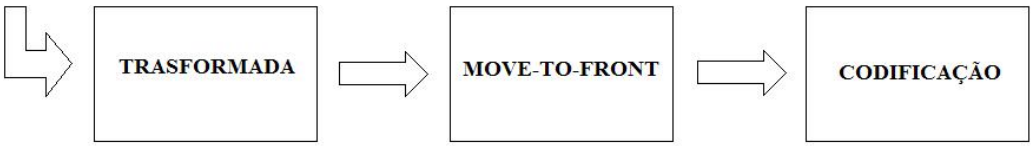


FIG. 5.1: Etapas do Algoritmo de Compressão *Block-Sorting*

A segunda etapa é a substituição dos valores de saída da transformação por números inteiros, de acordo com sua posição num alfabeto utilizado como apoio. Para esta etapa, são considerados algumas heurísticas para o problema de atualização de lista (19).

A terceira etapa é a codificação, que pode utilizar diversos tipos de compressores probabilísticos.

As explicações de detalhamento das etapas do BWTCA estão apresentadas nas sessões subsequentes deste capítulo e baseiam-se no documento (15), no qual foi divulgado o algoritmo *Block-Sorting*.

5.1 TRANSFORMADA DE BURROWS-WHEELER

A Transformada de Burrows-Wheeler ou BWT, trabalha sobre o texto de entrada permutando os caracteres de maneira que a ordenação original possa ser recuperada *a posteriori*, rapidamente e sem complicações.

O processo inicia criando uma matriz com permutações do texto original. Cada linha da matriz é preenchida com a linha predecessora deslocada ciclicamente em uma posição à esquerda. Se o texto fornecido de entrada tiver comprimento m, a matriz de permutação será do tipo m x m.

O passo seguinte é ordenar lexicograficamente as linhas e armazenando no arquivo final comprimido o valor do índice onde se encontra o texto original após a ordenação. A última coluna da matriz é retornada como saída do BWT e servirá de entrada para a próxima etapa, o *Move-To-Front*.

Para ilustrar a explicação, um exemplo prático retirado de (15) é apresentado. Seja s uma cadeia de caracteres de comprimento m pertencentes a um alfabeto $\sum$ de caracteres ordenados lexicograficamnte. $s =' abraca'$, $m = 6$ e $X =\{$ *'a','b','c','r'*$\}$.

Seja uma matriz M incializada como mostrado abaixo:

Linha	Cadeia
0	abraca
1	bracaa
2	racaab
3	acaabr
4	caabra
5	aabrac

Ordena-se M lexicograficamente (ordenação obedecida por X), obtendo:

row	string
0	aabrac
1	abraca
2	acaabr
3	bracaa
4	caabra
5	racaab

A coluna mais a direita, L, tem $'caraab'$ como valor de saída da transformada. O índice I tem o valor 1, que é onde a cadeia original s está localizada. Este índice deve ser armazenado de maneira que possa ser facilmente recuperável para a operação de reversão.

A BWT caracteriza uma etapa de transposição (confusão) de caracteres. Essa caracteristica pode ser aplicada em sistemas criptográficos.

A reversão do BWT é uma operação mais rápida que a codificação. Para obter o valor do texto original, sabe-se que o caractere da última posição de cada linha precede ciclicamente o caractere da primeira posição desta mesma linha, com isso já se possui a informação sobre quais são o primeiro e o último caracteres.

A partir da linha indicada pelo índice recuperado, verifica-se quantas ocorrências anteriores do caractere desta posição já aconteceram nesta última coluna, coluna **L**. Então sendo esta a k-ésima ocorrência, busca-se na primeira coluna (coluna **F**) a k-ésima ocorrência do mesmo caractere e se adiciona-o na última posição do texto original. O próximo passo é buscar o último caractere desta linha e também adicioná-lo na cadeia original, agora na penúltima posição e assim segue-se até concluir.

Como ilustração e para simplificar o entendimento apresenta-se a seguir a reversão do exemplo dado na codificação. Sejam $L =' caraab'$ e $i = 1$ a saída do **Move-to-Front** e o índice recuperado do arquivo descomprimido. Logo:

$i = 1 \Leftarrow L(1) =' A'$, 1^a ocorrência $\Leftarrow F(0)$.
'A' é adicionado a saída.
O último caractere da linha 0 é 'C'.
$L(0) =' C' \Leftarrow 1^a$ ocorrência $\Leftarrow F(4)$.
'C' adicionado a saída $\Leftarrow L(4) =' A'$
$L(4) =' A' \Leftarrow 3^a$ ocorrência $\Leftarrow F(2)$.
'A' adicionado a saída $\Leftarrow L(2) =' R'$.
$L(2) =' R' \Leftarrow 1^a$ ocorrência $\Leftarrow F(5)$.
'R' adicionado a saída $\Leftarrow L(5) =' B'$.
$L(5) =' B' \Leftarrow 1^a$ ocorrência $\Leftarrow F(3)$.
'B' adicionado a saída $\Leftarrow L(3) =' A'$.
$L(3) =' A' \Leftarrow 2^a$ ocorrência $\Leftarrow F(1)$.
'A' adicionado a saída. Como o processo retornou ao índice
de entrada, é verificado que o processo foi concluído. Saída: 'abraca'.

5.2 HEURÍSTICA *MOVE-TO-FRONT*

A heurística *Move-to-Front*, MTF, é utilizada para resolver o problema de atualização de listas (*List Update Problem*), LUP. Este problema consiste em fazer com que os itens mais consultados da lista estejam localizados nas primeiras posições da mesma, reduzindo seu custo de recuperação.

O MTF aplicado ao LUP simplesmente move o objeto recentemente lido para a primeira posição da lista, sem preocupar-se com os valores subsequentes a serem consultados. O MTF aplicado ao *Block-Sorting* tem o diferencial de retornar o valor onde o último item consultado estava, antes de ser deslocado para o ínicio da lista. Este valor é utilizado para substituir o caractere lido.

Para ilustrar a explanação sobre como o MTF trabalha, o mesmo exemplo utilizado para explicar o BWT será utilizado. Seja L a saída resultado do BWT, $L =' caraab'$, e $X =\{$ *'a','b','c','r'*$\}$. Passo-a-passo, tem-se:

Passo 1: 'c' é substituído por 2 e movido para a primeira posição, 0. X atualizado passa a ser $\{'c', 'a', 'b', 'r'\}$;
Passo 2: 'a' é substituído por 1 e movido para a primeira posição, 0. X atualizado passa a ser $\{'a', 'c', 'b', 'r'\}$;
Passo 3: 'r' é substituído por 3 e movido para a primeira posição, 0. X atualizado passa a ser $\{'r', 'a', 'c', 'b'\}$;
Passo 4: 'a' é substituído por 1 e movido para a primeira posição, 0. X atualizado passa a ser $\{'a', 'r', 'c', 'b'\}$;
Passo 5: 'a' é substituído por 0. Não ocorre atualização de X;
Passo 6: 'b' é substituído por 3 e movido para a primeira posição, 0. X atualizado passa a ser $\{'b', 'a', 'r', 'c'\}$;

Logo, a saída do MTF é $(2, 1, 3, 1, 0, 3)$. O MTF caracteriza uma operação de substituição (difusão) sobre os caracteres, mapeando-os em números inteiros. Assim como o BWT, essa característica pode ser utilizada num sistema criptográfico.

A reversão pra o MTF é simples e bastante similar ao processo de codificação. O primeiro número do vetor de inteiros obtido na saída do método compressor indica em qual posição do alfabeto está o primeiro caractere do texto original. Depois de lido, este caractere é trasportado para a primeira posição do alfabeto. O segundo número da cadeia indica a posição do segundo caractere do texto original, agora baseado no alfabeto recém modificado pelo transporte do primeiro. O restante do processo decorre da mesma maneira, até se ter descodificado todo o texto que serviu de entrada no processo de codificação.

5.3 POR QUE OCORRE MELHORIA NA COMPRESSÃO?

Considere as palavras '*that*' e '*what*'. Quando a lista de rotações for ordenada, as rotações iniciadas por 'hat' ficarão juntas. Uma grande parte das rotações serão terminadas por t e w, uma vez que são muito comuns na língua inglesa. A partir dessa permutação, a codificação pelo *Move-to-Front* substitui diversas seqüências de valores iguais por valores numéricos e se repetirão em maior escala, já que em um

momento uma repetição de 0's pode representar uma seqüência de 'a's, enquanto que em outro momento, pode representar uma seqüência de 'b's, alterando a freqüência (difundindo) dos símbolos, permitindo que compressores estatísticos, tal como o Huffman, obtenham melhores resultados.

t: hat acts like this:< 13 >< 10 >< 1
t: hat buffer to the constructor
t: hat corrupted the heap, or wo
w: hat goes up must come down< 13
t: hat happens, it isn't likely
w: hat if you want to dynamicall
t: hat indicates an error.< 13 >< 1
t: hat it removes arguments from
t: hat looks like this:< 13 >< 10 ><
t: hat looks something like this
t: hat looks something like this
t: hat once I detect the mangled

5.3.1 COMPARATIVO

A tabela 5.1 trás alguns resultados de compressão obtidos durante o desenvolvimento deste projeto. Em todos os casos o BSCA alcançou melhores resultados. Os arquivos utilizados fazem parte do Corpus Canterbury (50).

Arquivo/Algoritmo	Tamanho(Kb)	BSCA	ZIP	RAR	HUFFMAN
alice29.txt	152.089	45.228	54.476	51.260	87.794
asyoulik.txt	125.179	41.697	48.982	46.984	75.904
lcet10.txt	426.754	113.804	144.480	126.960	250.685
plrabn12.txt	481.861	154.347	195.042	176.035	275.701
bible.txt	4.047.392	846.370	1.190.092	979.503	2.218.541
world192.txt	2.473.400	447.910	724.514	531.314	1.558.729

TAB. 5.1: Comparando a BSCA com outros métodos compressores

6 MÉTODOS CRIPTO-COMPRESSORES

6.1 INTRODUÇÃO

A idéia de criar algoritmos criptográficos integrados a algoritmos de compressão de dados não é nova. O trabalho apresentado em (24) foi o pioneiro, mas pouca atenção foi dada a este tipo de pesquisa na década de 1990. A partir do ano 2000, diversos trabalhos foram apresentados (24; 28; 30; 26; 27; 25; 29; 37), mas apenas em (35) foi denominado o termo 'cripto-compressor' para definir algoritmos que implementam as funções de cifragem e compressão de dados simultâneamente.

A maioria destes trabalhos focou em modificações no Huffman e uns poucos em outros algoritmos, LZ e Codificação Aritmética e nenhum baseado no *Block-Sorting*, foco desta pesquisa.

Nos trabalhos que são apresentados a seguir o objetivo principal não é a garantia do sigilo absoluto para aplicações críticas, tais como segurança em transações bancárias ou militares, e sim apenas tornar a criptoanálise difícil a ponto de inviabilizar financeiramente o processo para se obter uma informação não-crítica.

6.2 TRABALHOS CORRELATOS

6.2.1 WAYNER

Uma árvore de Huffman é uma árvore ótima (36), mas existem várias outras árvores ótimas. Algumas delas são obtidas facilmente trocando-se de posição os símbolos de mesmo nível na árvore. Isto pode ser usado para se embaralhar a codificação. Usando esta característica, Wayner (24) propôs um método no qual é feita uma operação de ou-exclusivo (XOR) entre a chave secreta e os rótulos das arestas da árvore de Huffman. Esta operação equivale a usar uma outra árvore.

A operação de ou-exclusivo entre a chave e os rótulos da árvore seleciona os nós que farão parte da nova árvore, gerando um subconjuto de homofônicos, e a codificação então dá-se chaveando entre a árvore original e a gerada, utilizando a chave secreta

para tal. A cada leitura de caractere a ser codificado, o algoritmo verifica se o mesmo está na sub-árvore gerada. Caso positivo, o valor do bit da chave no momento da codificação é usado para decidir-se qual árvore será utilizada.

Como exemplo, tome a árvore de Huffman apresentada na figura 4.3 e seja $k = (0, 1, 1, 1, 0, 0, 1, 0, 1, 0)$ a chave secreta fornecida. A seleção dos nós dá-se através da equação $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) \oplus (0, 1, 1, 1, 0, 0, 1, 0, 1, 0) = (1, 2, 3, 6, 8)$, e portanto, a sub-árvore secreta será formada apenas pelos rótulos 1,2,3,6 e 8, que possuem freqüências 0.25, 0.15, 0.08, 0.05 e 0.05, respectivamente, como mostrado na figura 6.1.

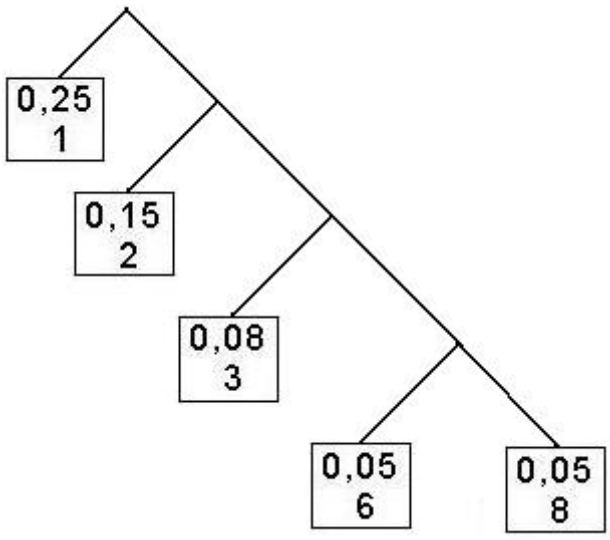


FIG. 6.1: Sub-árvore de Huffman pelo Algoritmo de Wayne

Uma limitação do algoritmo é o tamanho da chave. É necessário apenas um bit para cada rótulo da árvore de Huffman, e a quantidade de rótulos é a quantidade de caracteres distintos no texto a ser codificado, que é no máximo 256.

6.2.2 MILIDIÚ, MELLO, LUCENA E FERNANDES

O trabalho iniciado por Milidiú *et al.* em (25) renderam várias publicações em congressos (26; 27; 29; 31; 32; 33; 34), uma dissertação de mestrado (28) e uma tese de doutorado (35), totalizando 4 algoritmos cripto-compressores criados a partir das modificações do Algoritmo de Huffman Canônico.

6.2.2.1 INSERÇÃO SELETIVA DE NULOS

Este algoritmo foi apresentado em (25) e usa a técnica de esteganografia para esconder os símbolos codificados em símbolos falsos. É baseado na inserção seletiva de um número variável de símbolos nulos após os símbolos codificados. As perdas nas taxas de compressão são relativamente pequenas nesta abordagem.

São propostos dois procedimentos para agregar segurança ao Algoritmo de Huffman. O primeiro é baseado na cifra de substituição homofônica. São inseridos símbolos 'nulos' (símbolos sem significado) no texto cifrado. Sua inclusão tem por finalidade evitar uma fácil decodificação do texto por uma pessoa não autorizada. Esse símbolo 'nulo' é inserido no texto cifrado com múltiplos códigos, também chamados de *homofônicos*.

Esses homofônicos são gerados a partir da seguinte estratégia: após a construção da árvore de Huffman com N símbolos, assume-se que os primeiros m símbolos representam também os m códigos falsos. Nesta estratégia faz-se necessário a utilização de um bit identificador extra para indicar quando o código é verdadeiro (bit 0) ou falso (bit 1).

O segundo procedimento é aplicação de uma cifra de fluxo para a geração de uma cadeia de chaves criptográficas. A alternativa adotada é a *geração cíclica da cadeia de chaves* - quando a chave k termina o procedimento retorna ao início e assim por diante. Assim, o bit da chave é definido por $z_i = f(k, i) = k_{i \mod \phi}$, onde $\phi = |k|$ e mod é a operação de resto da divisão (módulo).

6.2.2.2 HUFFMAN HOMOFÔNICO-CANÔNICO

Este algoritmo apresentado em (27; 28) cria uma nova árvore homofônica baseada na árvore de Huffman canônica original para o texto de entrada.

Para construir este procedimento, usa-se os Códigos de Huffman Canônicos e a técnica criptográfica de Substituição Homofônica, associados às características de um Sistema Criptográfico de Chave Secreta. Na decodificação, associado com a Chave Secreta, é utilizado o algoritmo de decodificação dos Códigos de Huffman Canônicos.

Na codificação, utilizando o arquivo original, é feita uma varredura (*parsing*) para determinar as freqüências de cada símbolo. Logo após, são criados os símbolos homofônicos para os símbolos originais. O próximo passo é determinar os códigos canônicos de cada símbolo homofônico. É também nesta fase que é produzido o **Arquivo Modelo**, utilizado no processo de decodificação. A seguir, utilizando a chave secreta, é feita a permutação inicial dos símbolos homofônicos em cada nível da árvore. Como última etapa, temos a codificação do arquivo original, gerando o arquivo cifrado.

Na decodificação, utilizando as informações do Arquivo Modelo, é criada a tabela de decodificação. A configuração inicial da tabela de decodificação é baseada na permutação da chave secreta. Nesta etapa, também é feita a recriação dos símbolos homofônicos. Na última etapa, com o arquivo cifrado, é realizada a decodificação das informações.

6.2.2.3 HUFFMAN RANDOMIZADO

Apresentado em (31), este algoritmo é uma variante do algoritmo de Huffman que define um procedimento de cripto-compressão que aleatoriza a saída. O objetivo é gerar textos cifrados aleatórios como saída para obscurecer as redundâncias do texto original (confusão). O algoritmo possui uma função de permutação inicial, que dissipa a redundância do texto original pelo texto cifrado (difusão).

As alterações propostas são a inclusão de randomização aos dados comprimidos e a cifra por permutação inicial. É proposta uma menor perda na taxa de compressão através de uma quebra em blocos do texto e a transformação dos blocos em textos de tamanho uma potência de 2 para tornar a codificação perfeita.

São empregados os códigos Huffman canônicos com homofônicos de comprimento variável para codificar o texto original.

6.2.2.4 CÓDIGOS DE PREFIXO BASEADOS EM SUBSTITUIÇÃO HOMOFÔNICA COM 2 HOMOFÔNICOS

Este método cripto-compressor, denominado HSPC2, é o principal algoritmo dentre os criados por Milidiu et al. e foi apresentado em (32; 33; 34; 35). No processo de

codificação, o algoritmo adiciona um bit de sufixo em alguns códigos. Uma chave secreta e uma taxa de homofônicos são parâmetros que controlam essa inserção. Baseado nos valores destes dois parâmetros, o algoritmo seleciona que instância do símbolo recebe o bit sufixo. Isto pode ter diferentes textos cifrados para um mesmo texto claro e chave, devido ao método de substituição homofônica. O HSPC2 pode ser incluído em qualquer código de prefixo tais como os códigos Huffman padrão e Huffman Canônico (18).

Uma função de codificação é adicionada ao processo de codificação de prefixo, tornando difícil a quebra do código. Essa força criptográfica é analisada e o problema de decisão associado é reduzido ao problema do SUBSET-SUM (36) para demonstrar a NP-Completude como argumento da segurança do algoritmo. As funções de indexação e busca são mantidas.

É mostrado na tese que a quebra do HSPC2 é um problema NP-Completo.

6.2.3 KIM, WEN E VILLASENOR

Kim, Wen e Villasenor (37) publicaram recentemente um trabalho propondo outra modificação sobre a codificação aritmética, diferente do apresentado em (30), utilizando partição nos intervalos, baseada nos valores da chave criptográfica fornecida. O sistema consiste de uma primeira permutação aplicada à seqüência de entrada, e uma segunda permutação aplicada aos bits produzidos pelo codificador. A seqüência de valores da chave alimenta o gerador que provê informações a ambos os passos de permutação e para o divisor de intervalos do codificador aritmético.

A etapa de permutação deste método é similar à transformação ShiftRow do AES (48) e está apresentado na figura 7.2. A diferença entre os procedimentos é que no AES o deslocamento é realizado apenas para as linhas da matriz gerada e esse valor de deslocamento é fixo e predefinido para cada linha, enquanto que no trabalho de Kim *et al.*, o deslocamento é realizado tanto para as linhas como para as colunas da matriz e os valores de deslocamento são baseados na chave fornecida.

6.2.4 WANG

Em (30), o autor justifica a idéia de misturar as funções de compressão e criptografia porque ambas requerem muito tempo de processamento individualmente, podendo haver um ganho de desempenho com o entrelaçamento das operações. Para combinar os processos, Wang introduz a idéia de adicionar um embaralhamento randômico ao processo de compressão dos algoritmos.

O embaralhamento é definido da seguinte maneira: seja uma lista $(x_1, x_2, \ldots x_n)$ e se quer embaralhá-la randomicamente, então tem-se:

for i:= n downto 2 {
    k = random$(1, i)$;
    swap(x_i, x_k);
}

Importante observar que a permutação é baseada numa chave fornecida que serve de semente para o gerador de números pseudo-aleatórios.

6.2.4.1 PRIMEIRA MODIFICAÇÃO PROPOSTA - LZ *COMPRESSION*

A modificação nos compressores LZ dá-se a partir de uma permutação no valor inicial do dicionário. Numa implementação do tipo *codebook*, tal como uma compressão LZW, o dicionário consiste em seqüências de caracteres a ser processados. Inicialmente, o dicionário possui todas as seqüências de comprimento 1 listados em ordem alfabética. Neste caso, o embaralhamento é executado sobre todas estas seqüências e também sobre uma pequena quantidade de caracteres nulos que são gerados durante a execução da permutação. Na figura 6.2 tem-se um exemplo da inserção de nulos e do embaralhamento do dicionário. O acréscimo destes nulos tem por finalidade dificultar ataques do texto claro conhecido.

6.2.4.2 SEGUNDA MODIFICAÇÃO PROPOSTA - CODIFICAÇÃO ARITMÉTICA

Como já foi explicado anteriormente, o codificador aritmético trabalha comprimindo o texto num valor entre 0 e 1. Quanto maior o tamanho do texto a ser comprimido,

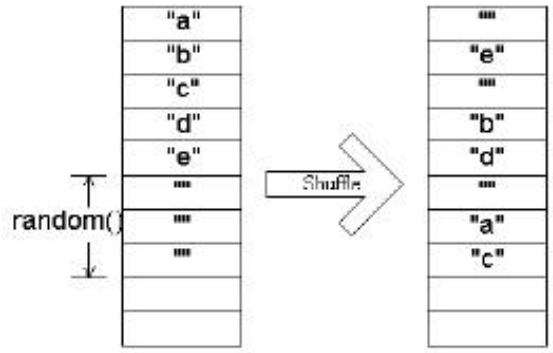


FIG. 6.2: Permutando Dicionário Inicial

maior a precisão da codificação. O algoritmo utiliza uma tabela de probabilidades para comprimir, dividindo o intervalo $[0, 1)$ em intervalos menores, de acordo com o próximo caractere da mensagem. Esse passo de subdivisão do intervalo é realizado até que toda a mensagem tenha sido codificada.

A proposta é utilizar a permutação pseudo-randômica para alterar a tabela de probabilidades. Por exemplo, se A tem probabilidade 0.24, B tem 0.12 e C tem 0.15, os intervalos originais iniciais seriam, respectivamente, $[0, 0.24)$, $[0.24, 0.36)$ e $[0.36, 51)$. Uma possível modificação seria $[0, 0.15)$, $[0.15, 0.27)$ e $[0.27, 0.51)$. É possível até melhorar a compressão em comparação ao original.

Esta abordagem é fraca contra ataques de texto claro conhecido. Para evitar tal fraqueza, foi proposto utilizar duas tabelas de probabilidade permutadas de maneiras diferentes. A divisão do intervalo de uma ou de outra tabela durante a codificação do caractere seria realizada baseada na chave fornecida. Se o bit da chave for 0 no momento da codificação, a primeira tabela é utilizada, caso contrário, utiliza-se a segunda.

6.2.4.3 TERCEIRA MODIFICAÇÃO PROPOSTA - HUFFMAN

A última modificação proposta em (30) é utilizando o algoritmo de Huffman adaptativo. Numerando os nós internos (nós com dois filhos) de maneira *top-down, left-right* por exemplo, cada nó no caminho percorrido até a folha que determina o caractere codificado no momento terá seus filhos permutados se o valor do bit da chave para aquele nó for 1. Para melhor compreensão, na figura 6.3, tem-se uma árvore de Huffman com os nós já numerados. A chave 101101 determina que os

nós 1, 3, 4 e 6 terão seus filhos permutados sempre que fizerem parte do caminho percorrido para a codificação do caractere.

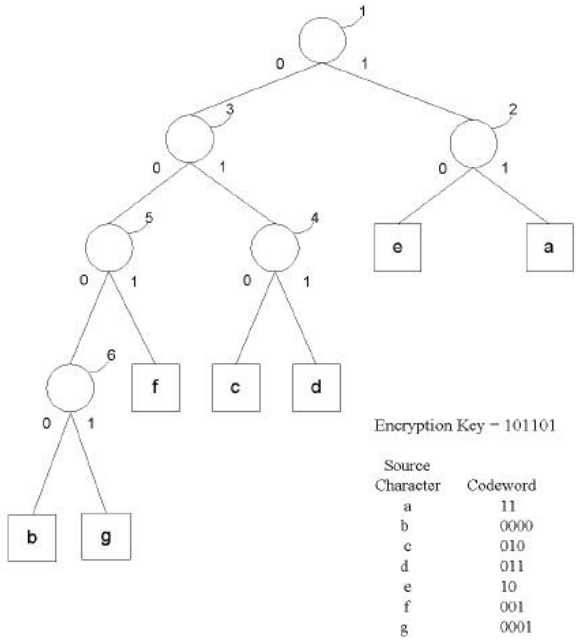


FIG. 6.3: Árvore de Huffman Permutada

Ao codificar o caractere 'a', os nós 1 e 2 são visitados. O nó 1 tem o primeiro bit da chave, o bit 1, relacionado a si, portanto depois da codificação de 'a', o nó 1 terá seus filhos permutados. O mesmo não ocorrerá ao nó 2, já que o segundo bit da chave é 0.

6.2.4.4 COMPRESSÃO

Wang afirma que não há perda na compressão para os métodos propostos. No primeiro caso, o dicionário modificado é praticamente o original, tendo apenas o acréscimo de alguns nulos. Na segunda proposta, os intervalos criados são os mesmos criados para uma versão sem modificações, com excessão da ordem em que ocorrem. Para a modificação do algoritmo de Huffman não há perda na compressão final, já que os nós são permutados num mesmo nível da árvore, ou seja, os nós permutados possuem o mesmo tamanho de codificação.

6.2.4.5 FORÇA CRIPTOGRÁFICA

Wang classifica os algoritmos apresentados como cifras de fluxo, *stream ciphers*. Para os primeiros dois métodos citados em (30), a etapa de permutação faz com que a força bruta para desfazer a permutação não dependa da chave, mas da quantidade de itens a serem permutados. Considerando que o dicionário do computador possui 256 caracteres, a criptanálise seria da ordem de 256!, o que equivale à quebra de uma chave de 1684 bits.

A segurança no Huffman modificado fica a cargo do tamanho da chave utilizada, que devido às restrições do tamanho da árvore, a chave pode assumir um comprimento máximo de 255 bits.

7 SISTEMA BZIP2S

7.1 INTRODUÇÃO

Neste capítulo serão apresentados as modificações no *Block-Sorting Compression*, os algoritmos *Keyed Move-to-Front*(**kMTF**) e o *Keyed Scrambling*(**kScr**). O kMTF é uma modificação no Move-to-Front para que seus deslocamentos no alfabeto sejam baseados no valor da chave fornecida. O kScr é uma cifra de transposição que atua sobre toda a saída do kMTF para dificultar a criptoanálise parcial do texto. Serão dados mais detalhes neste capítulo.

7.2 *MOVE-TO-FRONT* CRIPTOGRAFADO

Nesta sessão são propostas três alterações que acrescentam propriedades criptográficas ao algoritmo MTF.

Como já foi apresentado no capítulo 3 - Sistemas Criptográficos, um criptosistema é definido pela tupla (T, C, K, E, D) com as seguintes condições sendo observadas:

- Um espaço **T** de textos em claro;
- Um espaço **C** de textos cifrados, ou criptogramas;
- Um espaço **K** de chaves;
- Para cada chave k $\in$ **K**, $\mathbf{D}_k(\mathbf{E}_k(\mathrm{t})) = \mathrm{t}$.

7.2.1 CONSIDERAÇÕES PRELIMINARES

Antes de apresentar as modificações propostas neste trabalho de pesquisa, algumas considerações devem ser expostas para evidenciar as limitações de adicionar segurança ao algoritmo *Block-Sorting*. Estas limitações visam não alterar a característica principal da natureza proposta por Burrows e Wheeler ao criar a transformada: melhorar o resultado de métodos compressores, alterando a estatística dos caracteres do texto original. Ou seja, neste trabalho tem-se a intenção de não degradar o

resultado original do *Block-Sorting* a ponto de piorar o resultado final ao invés de melhorá-lo.

- Nenhuma modificação, permutação ou substituição, pode ser aplicada no texto claro fornecido como entrada e o operação BWT: O BWT trabalha sob o contexto do texto fornecido, assim como foi explicado ao justificar o porquê da boa compressão obtida com sua utilização. Efetuando qualquer operação criptográfica antes da execução do BWT perde-se a contextualização do texto claro, prejudicando o resultado final da compressão;
- Nenhuma modificação, permutação ou substituição, pode ocorrer entre a saída da operação BWT e a entrada do MTF: a lógica do resultado da saída do BWT deve ser preservado para que a transformação obtida com o MTF produza bons ganhos à compressão final. Caso a saída do BWT seja alterada, a característica de agregar sequencialmente caracteres iguais é perdida;
- Nenhuma operação de substituição deve ser efetuada sobre o resultado da saída do MTF modificado e a entrada do compressor. Caso o compressor utilizado seja o Huffman, pode-se operar uma cifra de transposição antes. Caso o compressor seja o Codificador *Run-Length*, a cifra de transposição só deve ser realizada após a compressão: O Block-Sorting traz ganhos para algoritmos de compressão que trabalham com estatística (contagem) dos caracteres. A repetição dos caracteres é o ganho da compressão. Caso a saída do MTF sofra operações de substituição, essa repetição será perdida.

7.2.2 ALTERAÇÃO NO DESLOCAMENTO

A primeira modificação proposta é efetuada sob o MTF. O novo algoritmo mantém o movimento de deslocamento em direção à primeira posição do alfabeto, mas não diretamente para a primeira. Este deslocamento é calculado a partir da posição atual do caractere lido e do valor corrente da chave, assim como apresentado nas equações 7.1 e 7.2.

$$NovaPosicao_i = (PosicaoAtual_i - Chave\,[i]) \tag{7.1}$$

$$NovaPosicao_i = NovaPosicao_i \mod (j+1) \tag{7.2}$$

A operação módulo $(j+1)$ é necessária para garantir que a nova posição do caractere seja maior que a atual. Por exemplo: se a posição atual for 3 e o valor da chave for 10, a nova posição seria 7, piorando a estatística de freqüência dos caracteres de maneira indesejada. Com a operação da segunda linha, a nova posição será 7 módulo 4, resultando em 3, que é o mesmo valor da antiga posição, não piorando o resultado e até, para este caso, incrementando a contagem do número 3. O motivo de o módulo ser $j+1$ e não j é para evitar divisão por zero, já que j e o valor da chave podem assumir esse valor simultaneamente.

O exemplo abaixo caracteriza a implementação proposta. Foi utilizado o mesmo exemplo do capítulo sobre o *Block-Sorting*, obtido após o texto claro ser processado pela Transformada (BWT), buscando facilitar o entendimento e apresentar a diferença gerada em relação ao algoritmo original.

Para o texto 'caraab', chave '120123' e alfabeto {*'a','b','c','r'*}, tem-se:

Passo 1: 'c' é substituído por 2. Com o valor da chave igual a 1,
a nova posição é 1. Alfabeto será {*'a','c','b','r'*};
Passo 2: 'a' é substituído por 0. Com o valor da chave igual a 2,
posição permanecerá 0. Alfabeto permanece {*'a','c','b','r'*};
Passo 3: 'r' é substituído por 3. Com o valor da chave igual a 0,
posição permanecerá 3. Alfabeto permanece {*'a','c','b','r'*};
Passo 4: 'a' é substituído por 0. Nenhuma atualização ocorre.
Passo 5: 'a' é substituído por 0. Nenhuma atualização ocorre.
Passo 6: 'b' é substituído por 2. Com o valor da chave igual a 3,
a nova posição é 1. Alfabeto será {*'a','b','c','r'*};

Esta abordagem introduz de maneira simples a influência da chave dentro do algoritmo *Move-to-Front*, transformando-o num algoritmo de cifra de fluxo, já que cada caractere é codificado individualmente, mas sendo influenciado pela codificação do caractere anterior, já que este tem influência no alfabeto.

O problema desta abordagem é que logo após um determinada quantidade de rodadas, o algoritmo passa a se comportar como o *Move-to-Front* não-alterado, característica não desejada. Vide 8, item 8.3, para detalhes sobre esta afirmação. A seguir serão propostas mais duas modificações que visam evitar que a convergência ao algoritmo original aconteça.

7.2.3 ALFABETOS PARALELOS PARA CODIFICAÇÃO

A segunda modificação proposta utiliza mais de um alfabeto dentro do *Move-to-Front*. A escolha do alfabeto a ser utilizado na codificação depende do valor da chave. Cada nova posição calculada é atualizada apenas no alfabeto corrente. A quantidade de alfabetos utilizados é parametrizável e são criados em tempo de execução. Além disso, eles devem diferir uns dos outros. Cada alfabeto criado deve sofrer uma permutação inicial baseado nas primeiras posições da chave fornecida. Como realizar essa permutação fica a cargo do programador da aplicação, e o importante é que essa permutação possa ser reproduzida fielmente na decodificação, caso a chave utilizada seja a correta. A implementação que será apresentada neste trabalho utiliza o SecureRandom da linguagem Java, que permite gerar cadeias pseudoaleatórias a partir de uma semente (valor base para gerar números). O SecureRandom implementa um gerador de números pseudoaleatórios baseado no algoritmo de resumo SHA1 (49). Vide pseudo-código em 7.2.3 para melhor compreensão.

Multiplos Alfabetos

```
1: for i = 1 to n do
2:    exec CriarAlfabetosEmbaralhados[n]
3: end for
4: for i = 1 to entrada.tamanho do
5:    byteAtual ⇐ entrada[i]
6:    alfabetoAtual ⇐ senha[i mod
      senha.tamanho] mod n
7:    for j = 0 to alfabeto[alfabetoAtual].tamanho − 1 do
8:       if alfabeto[currentAlphabet][j] = byteAtual then
9:          saida[i] ⇐ j
10:         novaPosicao ⇐ absoluto(j − (senha[i mod
            senha.tamanho]))
11:         novaPosicao ⇐ novaPosicao mod (j + 1)
12:         copia(alfabeto[alfabetoAtual], novaPosicao,
            alfabeto[alfabetoAtual], novaPosicao + 1, (j − novaPosicao))
13:         alfabeto[alfabetoAtual][novaPosicao] ⇐ byteAtual
14:         pare
15:      end if
16:   end for
17: end for
```

Esta abordagem torna o algoritmo mais dependente da chave criptográfica, que passa a influenciar em mais dois momentos, além daquele obtido com a primeira modificação: na permutação sobre os alfabetos e na seleção dos mesmos para efetuar a codificação e decodificação.

A utilização de múltiplos alfabetos traz como vantagem o aumento da quantidade de homofônicos de tamanho variável (vide definição de homofônicos de tamanho variável em (28)) que um caractere pode assumir, pois mesmo que este por acaso coincida em valor entre dois alfabetos durante a decodificação, o cálculo de sua nova posição modificará o alfabeto errado, fazendo com que todas as decodificações seguintes fiquem erradas, impossibilitando a criptanálise.

Infelizmente esta modificação piora o resultado da compressão (conforme apresentado no item 8.3), não resolve o problema destacado na primeira abordagem (a convergência ao algoritmo MTF original) e apenas retarda o fato. Depois de uma determinada quantidade de rodadas, o algoritmo passa a funcionar como um MTF comum. A diferença agora é o fato de utilizar mais de uma alfabeto. A solução do problema é apresentado no item 7.2.4.

7.2.4 PERMUTAÇÃO SOBRE OS ALFABETOS

A terceira e última modificação no MTF é a aplicação de uma decisão sobre realizar ou não uma permutação no alfabeto corrente, permutação semelhante à utilizada na modificação anterior, no momento da criação dos múltiplos alfabetos. A utilização desta permutação permite eliminar o problema de convergência do MTF criptografado ao MTF comum que não era eliminada com a aplicação das modificações anteriores.

Esta modificação é aplicada após a movimentação para frente do caractere recentemente lido. A partir do valor da chave, deve-se decidir se a permutação deve ou não ocorrer. Para esta decisão é utilizado um parâmetro percentual, indicando qual a probabilidade que a operação deve ocorrer. O valor que este percentual deve assumir para atingir uma boa segurança foi decidido através dos testes com o FIPS 140-2 (45) e está detalhado no capítulo 8, item 8.3.

A permutação ocorre a partir da posição seguinte àquela que o caractere recentemente lido foi transportado. Para melhor entendimento, vide figura 7.1.

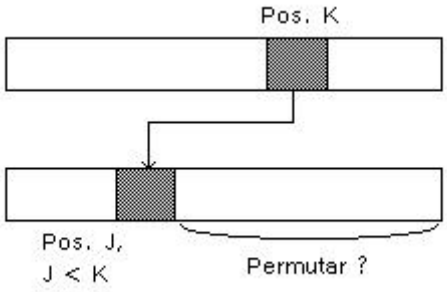


FIG. 7.1: Permutação sobre os Alfabetos

Na figura 7.1, um caractere na posição K é selecionado para movimentar-se no alfabeto, mudando para a posição J, tal que $J < K$. Então toma-se a decisão: permutar

ou não os elementos do vetor que vai de *J+1* ao final do alfabeto? Para tomar esta decisão, o algoritmo gera um número pseudo-aleatório a partir do valor corrente da chave. Se esse valor estiver dentro do intervalo de permutação, a operação é efetuada.

Esta abordagem dá um incremento criptográfico bastante interessante sobre o algoritmo, sem descaracterizar demais o MTF original. O caractere que aparece diversas vezes em seqüência será codificado de acordo com a primeira modificação. O caractere que tiver sido lido, logo antes do atual, poderá não estar mais na posição imediatamente anterior, caso o atual lhe tenha tomado a frente.

Para o exemplo abaixo, o parâmetro de permutação é 50%. Logo:

> Passo 1: 'c' é substituído por 2 e então movido para a posição 0.
> O alfabeto atualizado será {'c','a','b','r'};
> Passo 1.1: O valor da chave é 60, então executar permutação.
> A permutação será {'c','r','b','a'};
> Passo 2: 'a' é substituído por 3 e então movido para a posição 0.
> O alfabeto atualizado será {'a','c','r','b'};
> Passo 2.2: O valor da chave é 35, então a permutação não é executada.

7.3 TRANSPOSIÇÃO ANTES DA COMPRESSÃO

A única modificação proposta que não está incorporada internamente ao MTF é uma permutação antes da chamada do compressor, caso este seja o algoritmo de Huffman. Esta permutação é necessária para evitar que uma criptanálise parcial seja conseguida, permitindo revelar trechos do texto original, o que não é desejável. Diferentemente das permutações anteriores, que precisavam apenas ser reproduzidas novamente, esta necessita ter uma função inversa, já que este que é o último passo da codificação, e será o primeiro na decodificação, precisando desfazer a entropia por ele gerada. Neste caso, qualquer cifra de permutação reversível pode ser aplicada. Neste trabalho será apresentada apenas uma proposta de trasnposição reversível.

A sáida do MTF, modificado ou não, pode ser algo do tipo:

$$(5, 2, 7, 1, 0, 0, 0, 0, 6, 3, 4, 0, 0)$$

Isso pode ocorrer para o MTF Criptografado quando a chave contém uma ordenação que selecione o mesmo alfabeto de codificação da modificação anterior algumas vezes em seqüência. O problema consiste na seqüência de 0's, que no caso apresentado estão ligados ao número 1 que precede a seqüência. Caso o criptoanalista consiga decifrar qual caractere foi codificado como 1, todos os 0's corresponderão a este mesmo caractere, facilitando o trabalho do "atacante". Como é parte da natureza do Block-Sorting gerar um número excessivo de seqüências repetidas, a criptanálise de parte do texto dependerá apenas da decifragem de alguns números, fragilizando o sistema. Este é um ponto paradoxal neste cripto-sistema: quanto mais seqüências repetidas, maior a compressão final e menor é a segurança. Deve-se então criar um ponto de equilibrio que não interfira na compressão e tampouco na segurança: a aplicação da transposição reversível antes da compressão foi a solução encontrada. Com isso, a segurança ainda é reforçada, ou no mínimo, não prejudicada.

A transposição da saída do MTF modificado é eficiente porque desvincula as seqüências de número repetidos de seus antecedentes. Por exemplo, uma possível seqüência permutada da seqüência apresentada no parágrafo anterior seria:

$$(2, 4, 0, 7, 1, 0, 0, 3, 0, 5, 0, 6, 0,).$$

Apesar de ainda haver 0's em seqüência, não obrigatoriamente eles estarão ligados ao número imediatamente antecessor.

7.3.1 MODIFICAÇÃO

O módulo de transposição deste trabalho foi inspirado no ShiftRows do AES (48) e é também similar ao aplicado em (37).

A etapa ShiftRows do AES opera apenas sobre linhas da matriz de entrada; a operação desloca ciclicamente os bytes em cada linha em um certo valor (*offset*). Para o AES, a primeira linha é deixada inalterada. Cada byte da segunda linha é deslocado em uma posição à esquerda. Similarmente, a terceira e a quarta linha são deslocadas de dois e três respectivamente. Para blocos de tamanho 128 e 192 bits, o valor de deslocamento é o mesmo. Desta forma, cada coluna de saída sa etapa do ShiftRows é composta de bytes de outra coluna da entrada. Para o caso de

blocos de tamanho 256, a primeira linha não sofre deslocamentos, assim como nas situações anteriores. A segunda, terceira e quarta linhas são deslocadas em uma, três e quatro posições posições respectivamente - esta alteração para o bloco de 256 bits só é válida para o Rijndael (algoritmo original), já que o padrão AES não assume blocos de 256 bits. Como se pode notar, os deslocamentos são pré-determinados e em nenhum momento a chave secreta influencia as operações.

O trabalho de (37) diferencia-se do AES, pois não somente as linhas, mas também as colunas sofrem deslocamento cíclico. Um gerador cria chaves de tamanho 4 e de $\lceil N/4 \rceil$, onde N é o tamanho da entrada. As chaves de tamaho 4 são usadas para efetuar duas rodadas de deslocamentos intracolunas, intercaladas por uma operação de deslocamento interlinhas.

O *Key Scrambling*, ou kScr, gera uma matriz $[\lceil N/4 \rceil, 4]$, onde N é o tamanho do texto de entrada, preenchida com o resultado de saída do MTF criptografado. De modo semelhante ao trabalho de (37), o kScr realiza operações de deslocamento intralinhas e intracolunas, à esquerda e para cima respectivamente. Os valores destes deslocamentos são definidos pelo gerador de chaves alimentado por uma chave secreta. Aqui não ocorre uma segunda operação de deslocamento das posições da coluna.

A figura 7.2 e as matrizes 7.3 a seguir demonstram como funciona o kScr.

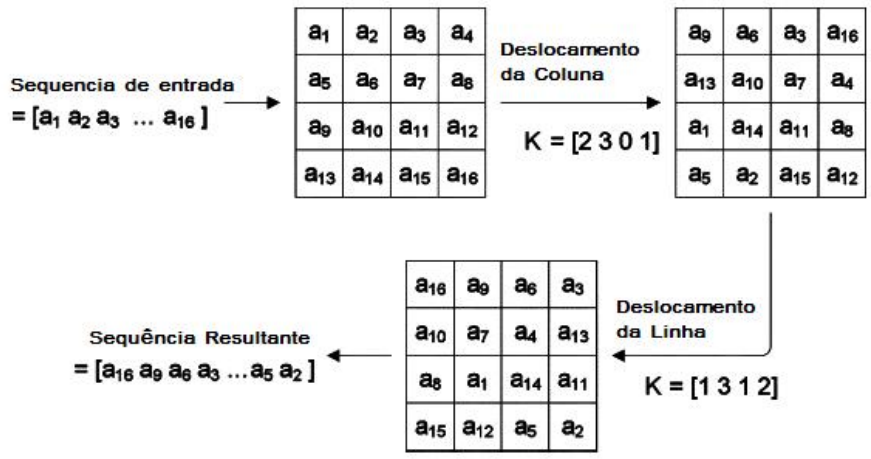


FIG. 7.2: Exemplo da Permutação KScr

$$\begin{pmatrix} c_1 & c_2 & c_3 & c_4 \\ c_5 & c_6 & c_7 & c_8 \\ c_9 & c_{10} & c_{11} & c_{12} \\ c_{13} & c_{14} & & \end{pmatrix} \overset{[2,3,0,1]}{\Longrightarrow} \begin{pmatrix} c_9 & c_6 & c_3 & c_{12} \\ c_{13} & c_{10} & c_7 & c_4 \\ c_1 & c_{14} & c_{11} & c_8 \\ c_5 & c_2 & & \end{pmatrix} \overset{[1,3,1,0]}{\Longrightarrow} \begin{pmatrix} c_{12} & c_9 & c_6 & c_3 \\ c_{10} & c_7 & c_4 & c_{13} \\ c_8 & c_1 & c_{14} & c_{11} \\ c_{13} & c_{14} & & \end{pmatrix} \tag{7.3}$$

Nas matrizes 7.3, a primeira flecha contém uma seqüência de valores que representa a chave a ser aplicada na permutação das colunas. Uma chave contendo 4 valores para operar 4 colunas. O primeiro valor, 2, deve ser aplicado à permutação cíclica, de cima para baixo, da primeira coluna a esquerda e assim por diante. Note que a matriz intermediária só deslocou os valores intracolunas.

A segunda flecha contém outra seqüência com valores da chave, que devem ser aplicados na operação de deslocamento das posições de cada linha.

Similar à permutação intracolunas, o primeiro valor da seqüência deve ser usado para executar o deslocamento cíclico na primeira linha da matriz, de cima para baixo.

7.3.2 CONSIDERAÇÕES SOBRE O KSCR

a) Qualquer permutação reversível, executada antes da aplicação da codificação pelo método de Huffman, não deve afetar o resultado final da compressão por não alterar a frequência dos caracteres do texto, mas apenas o posicionamento dos caracteres. Huffman atua contando os símbolos do texto, sem preocupar-se com a localização dos mesmos.

b) No caso do uso do *Run-Length Encoding* (RLE), onde o posicionamento dos caracteres importa, a abordagem deve ser outra: executar o RLE após o MTF criptografado e só depois aplicar a permutação. A conexão entre a seqüência de valores repetidos e seu predecessor ainda persistem após a aplicação do RLE, por isso a necessidade de utilizar o kScr.

7.4 GERAÇÃO DA CHAVE CRIPTOGRÁFICA

O BZip2s permite que qualquer valor de chave seja utilizado. O valor fornecido gera uma chave de criptografia de 32 bytes (256 bits) a partir de um método que fornece a senha como semente para um gerador de números pseudoaleatórios. Este método evita que chaves fracas sejam utilizadas diretamente no código. Caso não fosse utilizado tal gerador, uma senha com valores repetidos, tal como '0000000' atrapalharia a codificação pelo MTF Criptografado no momento do deslocamento e na permutação dos alfabetos.

Na implementação do cripto-sistema foi utilizado o gerador de números pseudoaleatórios da linguagem Java, o SecureRandom, já explicado na seção sobre o gerador de permutações neste capítulo, item 7.2.3.

Para mostrar como o gerador se comporta foi realizado um teste utilizando uma chave original de 32 bytes (256 bits) de valor 0 (zero). A cada rodada de um total de 256, um bit da chave era modificado e esse novo valor, a chave semente, alimentava o gerador SHA1 e a distância de Hamming era calculada entre a chave semente e a chave criada pelo gerador. O resultado é apresentado no gráfico 7.3. Nele pode ser notado que a posição do bit modificado não cria um padrão indesejável. Por exemplo, é indesejável que quando o bit modificado se encontra no primeiro terço da chave semente, as chaves geradas possuam sempre menor distância de Hamming que todas as chaves geradas a partir de sementes com o bit modificado pertencendo ao segundo terço. Além disso, todos os valores encontram-se na faixa de valor $[125, 147]$, ou $[48,8\%, 57,4\%]$, distribuídos aleatoriamente.

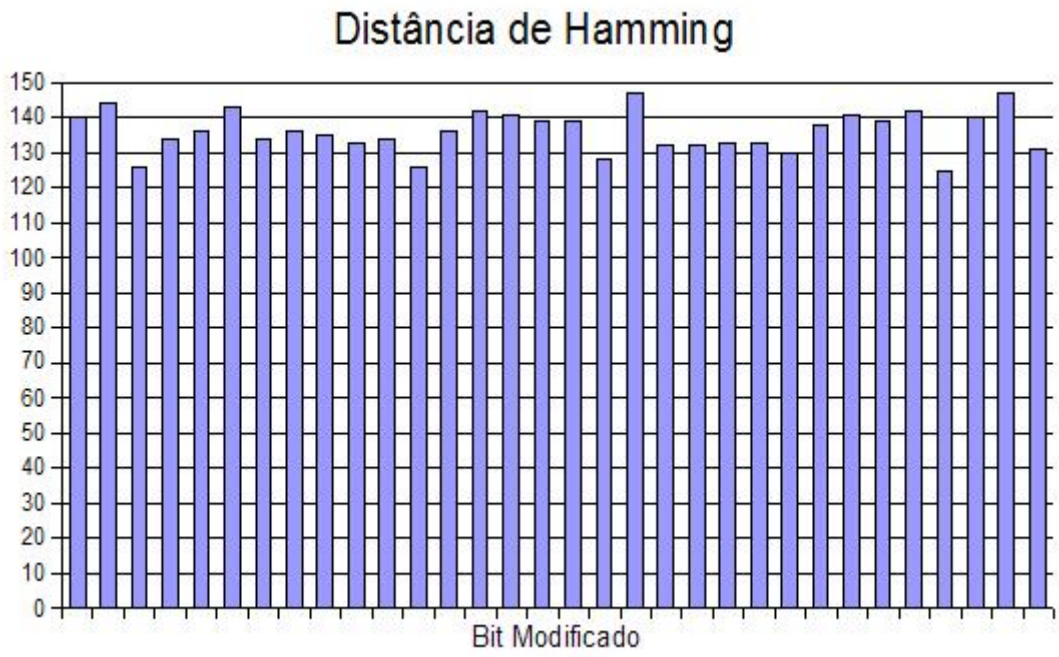


FIG. 7.3: Efeito do Deslocamento de Bits no Gerador de Chaves

8 SEGURANÇA DO CRIPTO-SISTEMA

8.1 CRIPTANÁLISE POR FORÇA-BRUTA

A segurança do *cripto-compressor* BZip2s pode ser avaliado dividindo-o por etapas: segurança do MTF Criptografado e segurança da permutação antes da compressão. Um sistema criptográfico não tem sua força reduzida se ocorre a aplicação de um outro método de cifragem independente.

8.2 SEGURANÇA DO MTF CRIPTOGRAFADO

Para cada passo realizado pelo MTF Criptografado (kMTF) para codificar os caracteres, o movimento avante do caractere é desconhecido, já que é baseado na chave gerada. É desconhecida qual a posição onde o caractere codificado no passo anterior e não se tem conhecimento também de como se comportou o restante do vetor alfabeto localizado após a nova posição calculada do caractere recém-codificado, já que pode ou não ocorrer a permutação deste sub-vetor e como essa permutação é gerada, já que é realizada por um gerador de números pseudoaleatórios baseados em SHA1.

De acordo com essas observações, monta-se a equação 8.2 que representa o esforço computacional necessário para efetuar uma criptoanálise por força-bruta sobre o kMTF. Em 8.2, m é o tamanho do texto criptografado e j é a posição desconhecida de caractere, codificada a cada rodada.

$$C = \prod_{i=1}^{m} \sum_{k=1}^{j} (255 - k)! \tag{8.1}$$

Para demonstrar quão elevado é o resultado deste cálculo e consequentemente, quão difícil é decodificar por força-bruta este criptosistema, será apresentado uma instância do problema. Seja 10 o valor médio para j e $m = 1$ *kbyte* (1024 bytes) o tamanho do texto claro fornecido de entrada.

$$C = \prod_{i=1}^{1024} \sum^{1} 0_{k=1}(255-k)! = \prod^{1024}(255\cdots254\ldots245!+254\cdots253\ldots245!+\ldots+245!) = 245!(255\cdots254\ldots \tag{8.2}$$

Sendo $9,36\cdots10^{516634} >>> 2^{256}$, a melhor estratégia para criptoanalisar o MTF Criptografado é atacando a chave criptográfica.

8.2.1 SEGURANÇA DA TRANSPOSIÇÃO ANTES DA COMPRESSÃO

Este tópico pode ser generalizado para qualquer tipo de algoritmo de cifragem por transposição. Como os dados apenas sofreram permuta de sua localização com outros e não tem sua natureza alterada, o ataque por força-bruta deve testar todas as combinações de posições possíveis.

Seja um texto cifrado de tamanho m. Para a primeira posição do texto o criptoanalista tem m opções, para a segunda $m-1$, para a terceira $m-2$ e assim por diante, até que ao chegar ao final do texto cifrado sobre apenas uma possibilidade. Ou seja, a quantidade de tentativas é $m!$.

Mesmo para um valor pequeno de m, um texto de 300 bytes por exemplo, a quantidade de variações que o método deverá testar é:

$$m! \Rightarrow \sqrt{2\cdot\pi\cdot n}\cdot(\tfrac{n}{e})^n \approx 8\cdots10^{613}, \text{ através da aproximação de Stirling (3).}$$

8.2.2 FORÇA DA CHAVE CRIPTOGRÁFICA

A tabela 8.1, é encontrada em (9) e apesar de bastante desatualizada, ainda serve para dar uma visão da relação custo x benefício relacionado o valor e o tempo gastos para quebrar uma chave criptográfica. Para valer a pena, a informação recuperada com esse esforço computacional tem que valer mais que o montande gasto para obtê-la.

A chave para o criptosistema proposto pode assumir qualquer valor que o usuário deseje, bastando apenas a atualização do parâmetro correspondente. Durante os

[1]Legenda para as estimativas de tempo:: ms: milisegundos, s: segundos, min: minutos, h: horas, d: dias, a: anos.

	Tamanho da Chave em Bits					
Custo	40	56	64	80	112	128
$100 K	2s	35h	1a	70.000a	$10^1$4a	$10^1$9a
$1 M	0.2s	3.5h	37d	7.000a	$10^1$3a	$10^1$8a
$10 M	0.02s	21min	4d	7000a	$10^1$2a	$10^1$7a
$100 M	2ms	2min	9h	70a	$10^1$1a	$10^1$6a

TAB. 8.1: Estimativa de Tempo Médio para Ataque de Força-Bruta por Hardware em 1995 [1]

testes que são apresentados a seguir, o gerador de chaves criou uma chave de 256 bits a partir de uma entrada de uma senha fornecida com 21 bytes (ou 168 bits). É importante que o usuário tenha em mente a tabela 8.1 quando seu desejo maior for a segurança e não a compressão.

Como já foi dito no capítulo 1, a idéia deste trabalho não é criar um novo sistema criptográfico comparável a sistemas consagrados, tais como o 3DES ou AES, mas fornecer um sistema seguro o suficiente para que o usuário doméstico sinta-se confortável ao comprimir seus arquivos pessoais no desktop de casa ou mesmo do trabalho e ainda obter um nível razoável de segurança sobre suas informações, apenas adicionando uma senha, mesmo sabendo que é possível que caso ela caia em domínio de terceiros e seja decodificada alguns meses depois. Esta facilidade não pode ser obtida utilizando soluções serializadas, já que o usuário tem o trabalho de executar as duas operações em seqüência.

Ano	Baixo	Médio	Alto
1990	398	515	1289
1995	405	542	1399
2000	422	572	1512
2005	439	602	1628
2010	455	631	1754
2015	472	661	1884
2020	489	677	2017

TAB. 8.2: Recomendações Otimistas de Rivest para o Tamanho da Chave (em *bits*)

Cruzando os valores nas tabelas 8.2 e 8.3, pode-se aferir uma estimativa de quais tamanhos de chave simétrica são recomendáveis para os dias atuais. Rivest estimou em (9) que no ano 2010, o sistema de chaves assimétricas deveria utilizar, uma chave

[1]Legenda para as estimativas de tempo:: ms: milisegundos, s: segundos, h: horas, d: dias, y: anos.

Tamanho da Chave Simétrica	Tamanho da Chave Assimétrica
56 bits	384 bits
64 bits	512 bits
80 bits	768 bits
112 bits	1792 bits
128 bits	2304 bits

TAB. 8.3: Comprimentos de Chave para Sistemas Simétricos e Assimétricos com Resistência Similar a Ataques de Força-Bruta

média de 631 bits, que é similar, em termos de resistência a ataques de força-bruta, a uma chave de 80 bits. Para um caso extremo, Rivest recomendou, naquela época, a utilização hoje de uma chava assimétrica com 1754 bits, que é aproximadamente similar a uma chave simétrica de 112 bits. Por tratar-se de recomendações feitas há mais de uma década, é plausível ser conservador e utilizar o sistema com uma chave de 128 bits, cujo espaço de busca por força-bruta é de 2^{128}, que é aproximadamente $3,4 \cdot 10^{38}$.

8.3 TESTES E RESULTADOS

Em (5) são apresentados alguns tipos de testes que são empregados para avaliar algoritmos criptográficos simétricos de bloco, tais como os testes divulgados no FIPS 140-2 (45), que serão descritos nas próximas subseções, testes de resistência contra criptanálises diferencial e linear e medidas de difusão.

8.3.1 TESTES DE ALEATORIEDADE

No FIPS 140-2 são especificados quatro testes de aleatoriedade de forma objetiva: *poker*, freqüência (*monobit*), seqüências corridas e seqüência longa. Estes testes são descritos a seguir, apresentando os limites de freqüências analisadas na estatística que são impostos para cada um deles. Estes testes não são especifícos para testes criptográficos. Na verdade, estes testes foram criados para análisar se um gerador de números pseudoaleatórios gera uma seqüência de bits de forma semelhante a uma saída realmente aleatória. Este tipo de resultado é desejável na criptográfia, e daí sua aplicação. As explicações a seguir foram retiradas de (5), com as atualizações para a versão 2 do FIPS.

8.3.1.1 TESTE DE FREQÜÊNCIA

Seja $S = S_0, S_1, S_2, S_3, ..., S_{N-1}$ uma seqüência binária de tamanho N. O objetivo deste teste é verificar se o número de bits iguais a 0 é aproximadamente igual ao número de bits iguais a 1, como é esperado em uma seqüência aleatória, já que a probabilidade de ocorrência de cada bit é $\frac{1}{2}$. Sejam N_0 o número de bits iguais a 0 e N_1 número de bits iguais a 1. A estatística utilizada é apresentada na equação 8.3:

$$X_1 = (N_0 - N_1)^2/N \tag{8.3}$$

Esta estatística deve seguir aproximadamente uma distribuição χ^2 (Chi-Quadrado) com grau de liberdade 1, para $N > 10$.

8.3.1.2 TESTE POKER

Seja m um inteiro positivo, tal que $\frac{n}{m} \geq 5 * (2^m)$, e seja k o maior inteiro menor ou igual a $\frac{n}{m}$. Divide-se a seqüência s em k blocos sem superposição de tamanho m e representa-se por n_i o número de ocorrências do i-ésimo tipo de seqüência de tamanho $m(1 \leq i \leq 2^m)$. O teste poker determina se cada seqüência de tamanho m aparece o mesmo número de vezes, como seria esperado em uma distribuição aleatória. A estatística utilizada é a apresentada a seguir:

$$X_3 = 2^m k(\sum_{i=1}^{2^m} N^2) - k \tag{8.4}$$

Assim como para o teste anterior, esta estatística deve seguir aproximadamente uma distribuição χ^2 com $(2^m - 1)$ graus de liberdade. Esse teste é uma generalização de teste de freqüência.

8.3.2 TESTE DE SEQÜÊNCIAS CORRIDAS

O objetivo deste teste é verificar se o número de ocorrências de seqüências compostas somente de bits iguais a 0, ou seja, delimitadas à esquerda e à direita por 1, ou somente bits iguais a 1, delimitadas à esquerda e à direita por 0, é compatível com uma distribuição aleatória.

O número esperado de tais seqüências de comprimento i em uma seqüência de tamanho n é $e_i = (n - i + 3)/2^{i+2}$. Seja k o maior inteiro i para o qual $e_i \geq 5$. Sejam U_i e Z_i os números de seqüências de 1's e 0's de tamanho i presentes em n, respectivamente, para cada i, $(1 \leq i \leq k)$. A estatística usada é apresentada na equação 8.5.

$$X = \sum_{i=1}^{k} \frac{(U - e)^2}{e_i} + \sum_{i=1}^{k} \frac{(Z - e)^2}{e_i} \tag{8.5}$$

Esta estatística deve seguir aproximadamente uma distribuição χ^2 com $2k-2$ graus de liberdade.

8.3.3 PARÂMETROS DEFINIDOS NO FIPS 140-2

Uma seqüência s de bits de tamanho 20.000, saída do gerador de aleatórios a ser analisado, está sujeita a cada um dos seguintes testes. Se o gerador for reprovado em pelo menos um dos testes, este gerador é rejeitado. A seguir são apresentados os parâmetros limítrofes para cada teste.

a) **Teste de freqüências:** O número n_1 de bits iguais a 1 em s deve obedecer o intervalo $9725 < n_1 < 10257$.

b) **Teste Poker:** A estatística X_3 definida na equação 8.4 é computada para $m = 4$. O gerador é aprovado caso seja obedecido o intervalo $2.16 < X_3 < 46.16$.

c) **Teste de Seqüência Longa:** O gerador de aleatórios é aprovado caso nenhuma seqüência de bits iguais possua tamanho maior que 27.

d) **Teste de Seqüências Corridas:** Os números U_i e Z_i de tamanho i são contados para cada i no intervalo $1 \leq i \leq 6$. As seqüências de tamanho maior que 6 são consideradas na contabilização das seqüências de tamanho 6. O gerador é aprovado caso as contagens para U_i e Z_i obedeçam os limites apresentados na tabela 8.4.

Tamanho da Seqüência	Valores Permitidos
1	2343 - 2657
2	1135 - 1365
3	542 - 708
4	251 - 373
5	111 - 201
6	111 - 201

TAB. 8.4: Valores para Teste de Seqüências Corridas

8.4 RESULTADOS OBTIDOS PARA O BZIP2S

O BZips foi implementado na linguagem Java, compilador jdk 1.6.0_03, IDE Eclipe, utilizando um notebook HP com processador Celeron M 1.7Mhz e 1Gb de Mémoria DDR. Para a comparação não deixar dúvidas quanto aos resultados comparativos, o Huffman utilizado foi para gerar este resultados foi o mesmo aplicado internamente no código do BZip2s.

8.4.1 TESTES DE ALEATORIEDADE

8.4.1.1 RESULTADO DO TESTE FIPS 140-2

Após várias rodadas de testes variando o valor de permutação e a quantidade de alfabetos, para uma taxa de permutação menor que 40% e qualquer quantidade de alfabetos auxiliares, o BZip2s foi reprovado no NIST FIPS 140-2. Baseado nestes resultados, definiu-se que estes devem ser os requisitos mínimos de segurança a serem aplicados no cripto-sistema proposto.

Na seção de testes de compressão foi apresentados gráficos e tabelas que mostram a influência de ambos os paramêtros (valor de permutação e quantidade de alfabetos) no resultado final da compressão e também no desempenho de compressão.

A implementação dos testes NIST foi retirada de (5), atualizada para os parâmetros do FIPS 140-2, já que o original encontra-se com os parâmetros para o FIPS 140-1.

Tamanho do Texto Cifrado: 20000 bits (ou 2500 bytes).

a) **Teste Monobit:** RESULTADO:n1=9947. APROVADO!

b) **Teste Poker:** RESULTADO:X3=22,489599999999882. APROVADO!

c) **Teste de Seqüências Longas:**

Maior Seqüência de 0

Valores permitidos: até 27. RESULTADO: 12. APROVADO!

Maior seqüência de 1

Valores permitidos: até 27. RESULTADO: 12. APROVADO!

d) **Teste de Seqüências Corridas**

Teste de Seqüências de 0

Tamanho 1: 2427 ocorrênciass. RESULTADO: APROVADO!
Tamanho 2: 1185 ocorrênciass. RESULTADO: APROVADO!
Tamanho 3: 654 ocorrênciass. RESULTADO: APROVADO!
Tamanho 4: 319 ocorrênciass. RESULTADO: APROVADO!
Tamanho 5: 154 ocorrênciass. RESULTADO: APROVADO!
Tamanho 6: 175 ocorrênciass. RESULTADO: APROVADO!

Teste de Seqüências de 1

Tamanho 1: 2398 ocorrênciass. RESULTADO: APROVADO!
Tamanho 2: 1235 ocorrênciass. RESULTADO: APROVADO!
Tamanho 3: 634 ocorrênciass. RESULTADO: APROVADO!
Tamanho 4: 332 ocorrênciass. RESULTADO: APROVADO!
Tamanho 5: 162 ocorrênciass. RESULTADO: APROVADO!
Tamanho 6: 152 ocorrênciass. RESULTADO: APROVADO!

O resultado obtido é interessante para o cripto-sistema, já que o mesmo não efetua rodadas para misturar mais os dados, como a maioria dos algoritmos tradicionais. A execução de diversas rodadas facilita o embaralhamento da informação e, conseqüentemente, dando mais aleatoriedade a saída. Este resultado garante que a codificação

do BZip2s tem um comportamento estatisticamente idêntico ao de um gerador de números aleatórios, caracteristica desejável para um algoritmo de criptografia.

8.4.2 TESTES COMPARATIVOS DE COMPRESSÃO

A tabela 8.5 traz resultados da compressão de arquivos texto do Canterbury Corpus(50). Foram utilizados os compressores *Block-Sorting* original (BWT-MTF-HUFFMAN), o BZip2s, apresentado neste trabalho, e o Código de Huffman.

Pode-se observar que para arquivos pequenos (<200Kb), a compressão pelo BZip2s tem resultado pior que o obtido pelo método de Huffman, demonstrando que para estes casos o BZip2s não é adequado ao uso. Para os demais testes realizados, a compressão por BZip2s mostrou-se bastante satisfatória para o caso de segurança mínima definida a partir dos resultado do FIPS 140-2: 1 alfabeto auxiliar e 40% de permutação sobre este alfabeto.

A curva do gráfico 8.1 mostra que há uma tendência de o BZip2s comprimir melhor que o Huffman para maiores quantidades de informação, enquanto sua perda em relação ao BSCA tende a ficar praticamente estável.

Senha utilizada: 'DANIELTESTANDOSISTEMA'.

Arquivo/Algoritmo	Tamanho(Kb)	BSCA	BZIP2S[1]	HUFFMAN
asyoulik.txt	125.179	41.697	79.399	75.904
alice29.txt	152.089	45.228	87.361	87.894
lcet10.txt	426.754	113.804	217.872	250.685
plrabn12.txt	481.861	154.347	293.222	275.701
world192.txt	2.473.400	447.910	844.282	1.558.729
bible.txt	4.047.392	846.370	1.806.114	2.218.541

TAB. 8.5: Tabela Comparativa de Desempenho de Algoritmos de Compressão

O gráfico da figura 8.2 é referente a tabela 8.6 e mostra como o percentual de permutação sob o alfabeto e a quantidade de alfabetos auxiliares do MTF Criptografado influenciam no resultado final da compressão. Pode-se verificar que para o caso do arquivo 'asyoulik.txt', que possui 123Kb (125.179 bytes), a introdução da permutação, mesmo que de apenas 20%, piora o resultado em 73%. A partir daí, a

[1]Foi utilizado o BZip2s com 40% de Permutação e 1 Alfabeto.

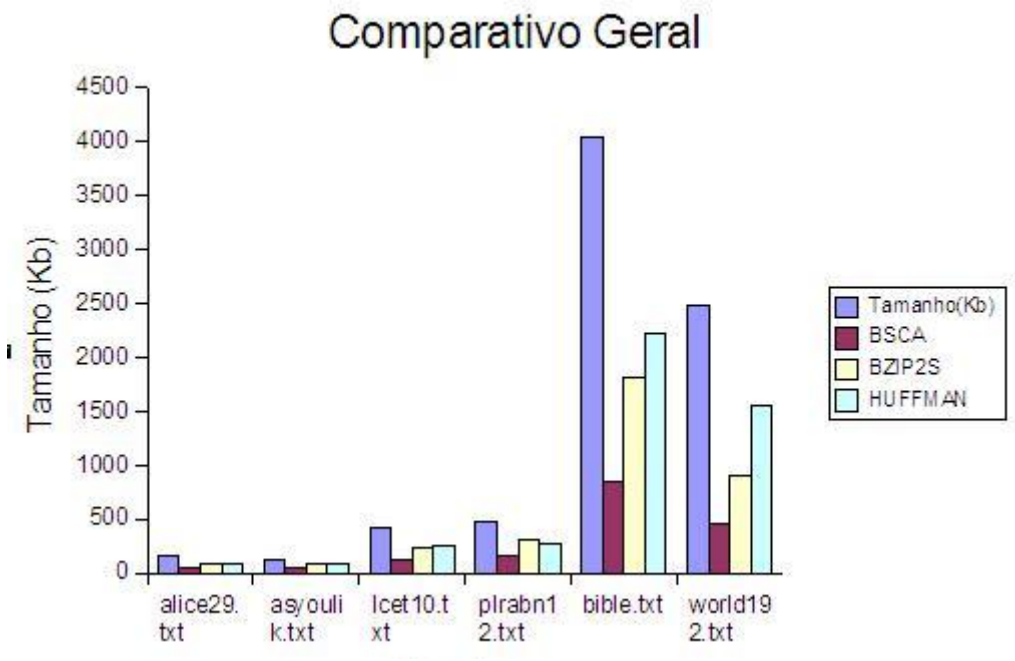


FIG. 8.1: Gráfico Comparativo de Desempenho de Algoritmos de Compressão

asyoulik.txt	0%	20%	40%	70%	100%
Tamanho(bytes)	125.179				
BSCA	41.697				
HUFFMAN	75.904				
BZIP2S a1	42.954	74.276	79.399	82.712	86.764
BZIP2S a2	49.821	82.483	89.915	94.325	98.564
BZIP2S a3	50.984	83.168	90.156	93.838	99.243

TAB. 8.6: Tabela Comparativa de Desempenho de Compressão para o Arquivo "Alice29.txt" (50)

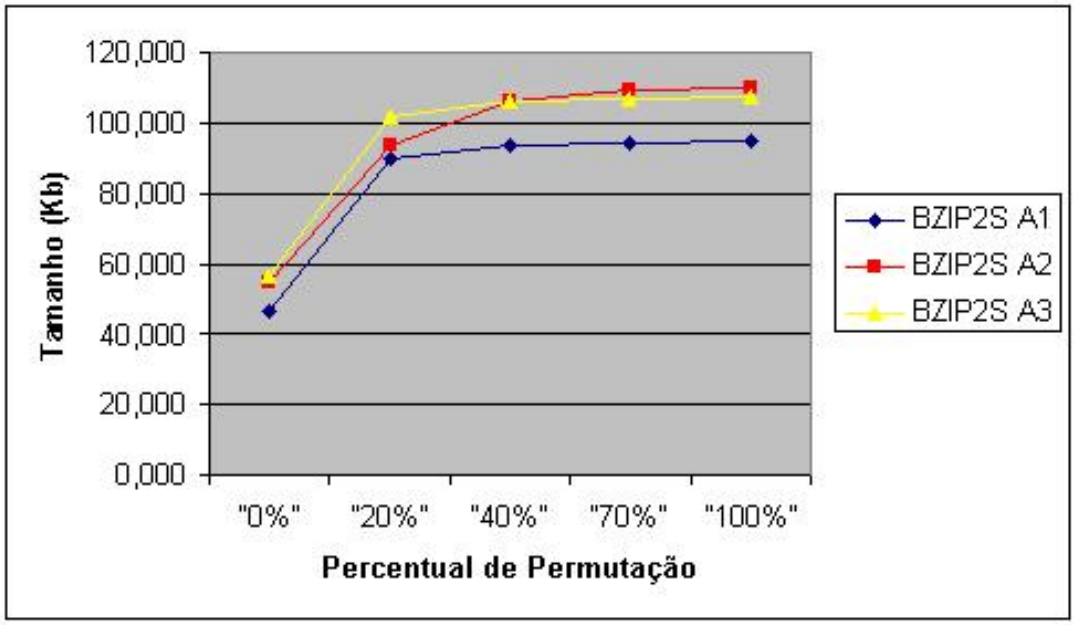


FIG. 8.2: Gráfico para Demonstração de Influências Permutação/Alfabetos

variação entre os valores de permutação, de 20% até 100%, degrada o resultado em apenas em 33,6%, do melhor para o pior resultado.

A influência da quantidade de alfabetos é evidentemente menor. Analisando apenas os valores de uma mesma coluna, vê-se pouca variação entre os resultados, com 18% sendo a maior diferença, acontecendo na coluna de permutação 0%.

Para dar mais sustentação aos resultados apresentados àcima, foi elaborada uma tabela similar para o maior arquivo da tabela 8.5. Aqui, assim como para o caso anterior, vê-se que a influência da permutação também é elevada. A variação entre as permutações 0% e 20% para 1 alfabeto foi de 78%. Já entre os valores com permutação entre 20% e 100%, entre o melhor e o pior resultado, tem-se uma variação de 53%, relativamente maior que a variação para o menor arquivo.

Já a diferença percentual entre a quantidade de alfabetos auxiliares, tem-se como maior variação 27,2%, obtido também na coluna de permutação 0, assim como para o primeiro caso em 8.6.

Nota-se em 8.7 que para todos os casos, a compressão com 3 alfabetos obteve um resultado melhor que a compressão por 2 alfabetos. Isso se deve ao valor da chave selecionada para teste. Os valores escolhidos selecionaram um mesmo alfabeto mais vezes que os outros, resultando num incremento da compressão. Para o arquivo

bible.txt	0%	20%	40%	70%	100%
Tamanho(bytes)	4.047.392				
BSCA	846.370				
HUFFMAN	2.218.541				
BZIP2S a1	857.146	1.525.906	1.654.489	1.753.688	1.875.996
BZIP2S a2	1.091.006	1.811.226	2.022.671	2.147.237	2.340.786
BZIP2S a3	1.076.366	1.794.013	1.977.083	2.089.992	2.262.122

TAB. 8.7: Tabela Comparativa Para o Arquivo 'bible.txt'

menor, apresentada na tabela 8.6, em apenas um caso a compressão de 2 para 3 alfabeto foi melhorada. Esse fato mostra a influência da Transformada de Burrows-Wheeler no processo, que chaveia a entrada pelo seu contexto, já que em todos os testes foi utilizado a mesma chave.

bible.txt	0%	20%	40%	70%	100%
HUFFMAN	262,12%				
BZIP2S a1	101,27%	180,29%	195,48%	207,20%	221,65%
BZIP2S a2	128,90%	214,00%	238,98%	253,70%	276,57%
BZIP2S a3	127,17%	211,97%	233,60%	246,94%	267,27%

TAB. 8.8: Comparação Percentual com a Compressão BSCA

O gráfico 8.3 apresenta o resultado condensado da tabela 8.8 em percentuais referentes ao *Block-Sorting* original, ou seja, o gráfico mostra a perda de compressão, deixando visivel que até uma permutação de 70%, para quantidades de até 3 alfabetos, o resultado do Huffman é melhorado, mesmo com a criptografia, alcançando nosso objetivo principal: prover segurança sem descaracterizar o *Algoritmo de Compressão pela Transformada de Burrows-Wheeler*, o BWTCA.

8.4.3 JUSTIFICANDO AS PERMUTAÇÕES NOS ALFABETOS AUXILIARES

A força criptográfica do BZip2s depende praticamente da decisão de permutar ou não o alfabeto auxiliar no último passo de cada rodada do MTF Criptografado (kMTF). Para demonstrar essa afirmação foram realizadas modificações no *Block-Sorting* e no MTF Criptografado de modo que fossem executados apenas os passos 1 e 2 de ambos os algoritmos.

Foi implementado um outro programa que lê tais saídas e conta quantos bytes coincidiram na mesma posição. O teste foi executado para o arquivo 'alice29.txt',

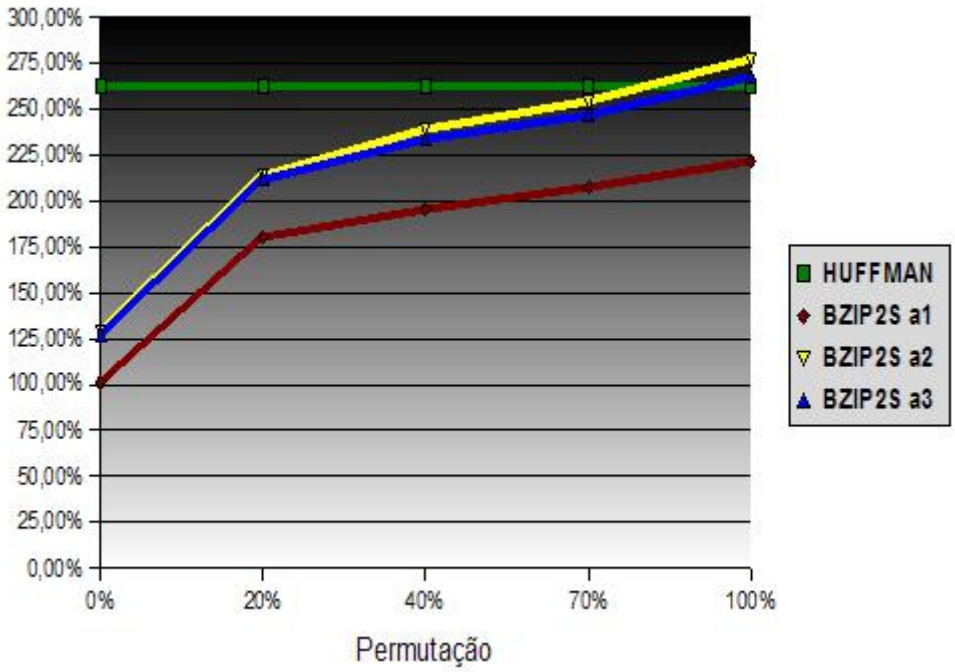


FIG. 8.3: Gráfico Percentual em Relação à Compressão BSCA

que possui 152.089 bytes, para os valores de permutação 0% e 40%, com 1 e 3 alfabetos e são apresentados na tabela 8.9. O valor em cada célula é o percentual de bytes coincidentes, posição a posição, ou $[1 - H_d(Original, Criptografado)]$.

Deve-se destacar que para o primeiro caso da tabela, 0% de permutação para 1 alfabeto, os últimos 27347 bytes coincidiram em 81% dos casos, mostrando que sem a aplicação da permutação, o kMTF trabalharia praticamente como o *Move-to-Front* original, confirmando a afirmação feita no capítulo sobre o MTF Criptografado.

Qtd. Alfabetos	0%	40%
1	61%	38,5%
3	41,5%	27,4%

TAB. 8.9: Valores para o Teste de Variação Byte-a-Byte

9 CONSIDERAÇÕES FINAIS

Este trabalho teve início com a necessidade de estender a pesquisa realizada em (35), buscando desenvolver novos sistemas baseados em algoritmos de compressão promissores, neste caso o *Block-Sorting Compression Algorithm*. Além disso, a necessidade de descobrir novos métodos criptográficos e também de melhorar os resultados de compressão para sistemas criptocompressores incentivaram o desenvolvimento deste trabalho.

O foco deste trabalho, então, foi a criação de um algoritmo criptocompressor, o BZip2s, baseado na compressão *Block-Sorting*. Foi utilizado neste trabalho o termo cripto-compressor, apresentado em (35), para referenciar algoritmos que implementem as funções de cifragem e compressão de dados simultaneamente.

O objetivo foi criar um cripto-compressor que baseado em algoritmos de compressão de dados, sobre os quais se embute a segurança. Este objetivo foi alcançado. Também foi objetivado que o cripto-compressor desenvolvido apresentasse bons resultados em comparação a métodos compressores puros, tanto no quesito de desempenho de compressão quanto no desempenho de tempo. Neste caso, apenas o quesito de desempenho de compressão foi alcançado. As modificações realizadas no *Block-Sorting* se mostraram bastante custosas, como mostram os resultados apresentados no capítulo 8.

Foram analisados várias técnicas criptográficas: monoalfabéticos, polialfabéticos, homofonia, esteganografia, permutação, entre outras, e como elas poderiam ser aplicadas para incluir segurança em algoritmos de compressão de dados. Foram analisadas diversos sistemas criptográficos simétricos (DES, AES, IDEA, RC5, RC6. etc), de resumo da mensagem para a geração de chaves(MD5, SHA1, etc.), arquiteturas como redes de Feistel, redes de substituição-permutação (SPN), caixas de substituição, entre outros, de tal forma a dar subsídios à criação de sistemas criptográficos próprios. As técnicas de criptoanálise diferencial e linear foram estudadas para permitir que fossem feitas auto-análises das propostas de segurança. Também foram estudados outras heurísticas para solucionar o problema de atualização de

lista para o qual o *Move-to-Front* se propõe.

Foram feitas então várias tentativas de criação de métodos para modificar a Transformada de Burrows-Wheeler e a heurística *Move-to-Front*, e depois de definido que as modificações só seriam válidas sobre o MTF, as codificações mais promissoras foram registradas nesta dissertação.

Para a avaliação da segurança dos métodos foram feitos vários experimentos matemáticos e práticos. A ferramenta do *National Institute of Standards and Technology* (**NIST**), '*A statistical test suite for random and pseudorandom number generators for cryptographic applications*' (45), foi muito útil para encontrar a taxa de permutação mínima necessária sobre o alfabeto do MTF Criptografado para aleatorizar a saída do cripto-compressor, além de fornecer uma garantia de que, aparentemente, o cripto-compressor proposto é confiável.

No estudo de pesquisas correlatas, alguns artigos foram encontrados. Em nenhum deles porém há notícias de algum algoritmo cripto-compressor baseado na Transformada de Burrows-Wheeler e no *Move-to-Front* ou mesmo sendo explorado comercialmente. Sistemas operacionais como o Windows e ferramentas populares, quando apresentam, trazem soluções serializadas.

De modo geral, o algoritmo BZip2s apresenta uma expansão considerável na compressão do arquivo, se comparado com o algoritmo de compressão original, quando se utiliza a taxa mínima de permutação sobre o alfabeto do MTF Criptografado, mas ainda mantendo um incremento do resultado da compressão por Huffman. Uma pequena sobrecarga no desempenho de compressão e a manutenção da velocidade de descompressão, sendo assim voltados para a aplicação a que se propõe.

9.1 TRABALHOS FUTUROS

Este trabalho pode ser continuado pesquisando a criação de cripto-compressores baseados nos diversos algoritmos compressores existentes, tendo sido ou não foco de outras pesquisas. Outra oportunidade seria implementar os trabalhos correlatos citados nesta dissertação e que não apresentam análises teóricas, implementações e resultados, o que permitiria que fossem efetuadas comparações entre os sistemas.

As técnicas de criptoanálise existentes e explicadas no capítulo 3, item 3.2.5, são

específicas para sistemas criptográficos simétricos e não produzem resultados para algoritmos cripto-compressores. Seria interessante o estudo e desenvolvimento de técnicas de criptoanálise voltados para esta nova natureza de algoritmos.

10 REFERÊNCIAS BIBLIOGRÁFICAS

[1] Feistel, H., Cryptography and Computer Privacy, Scientific American, May, p.15-23, 1973.

[2] Mao, Wembo. Modern Cryptography: Theory Practice. Prentice-Hall, 1st Edition, 2003.

[3] Garret, Paul. The Mathematics of Coding Theory. Pearson Prentice-Hall, 1st Edition, 2004.

[4] Stallings, William, Cryptography and Network Security: Principles and Practice, Prentice-Hall, 2nd Edition, 1999.

[5] Lambert, Jorge, Cifrador Simétrico de Blocos: Projeto e Avaliação, Dissertação de Mestrado, IME, 2004.

[6] Terada, Routo, Segurança de Dados: Criptografia em Redes de Computador, Ed. Edgard-Blucher, 1a Edição, 2000.

[7] Shannon, C., A Mathematical Theory of Communication. Bell System Technical Journal, vol. 27, no. 3, pp. 379-423, 1948.

[8] Shannon, C., Communication Theory of Secrecy Systems, Bell System Technical Journal, vol. 28, no. 4, pp. 656-715, 1949.

[9] Schneier, Bruce. Applied Cryptography: Protocols, Algoritms, and Source Code in C. Wilwy, 2nd Edition, 1996.

[10] Singh, Simon, O Livro dos Códigos, Ed. Record, 1a Edição, 2001.

[11] Szwarcfiter, Jayme e Markenson, Lilian. Estruturas de Dados e Seus Algoritmos. Editora LTC, 2a edição, 1994.

[12] Matsui, M., Linear Cryptanalysis Method for DES Cipher, Advances in Cryptology Eurocrypt93 Proceedings, Springer Verlag, New York, pág. 386-397, 1994.

[13] Matsui, M., The First Experimental Cryptanalisys of the Data Encryption Standard, Advances in Cryptology CRYPTO94 Proceedings, Springer Verlag, New York, pág. 1-11, 1994.

[14] Mariño,F.C.C., Análise da Segurança das Cifras Rijndael e Serpent Contra as Criptoanálises Linear e Diferencial. Teste de Mestrado, IME, Rio de Janeiro, 1998.

[15] Burrows, M., Wheeler, D., A block-sorting lossless data compression algorithm, Digital Tech Report number 124 (1994).

[16] Salomon, David. Data Compression: The Complete Reference. Springer, 2a edição, 2000.

[17] Ziv, J. e Lempel, A. A universal Algorithm for Data Compression. IEEE Transactions on Information Theory, IT-23(3):337-343, 1977.

[18] Ian H. Witten and Alistair Moffat and Timothy C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann Publishers. San Francisco, CA, isbn 1-55860-570-3, 1999.

[19] Christoph Ambühl. On the List Update Problem. Doctoral Dissertation, Swiss Federal Institute of Techology, 2002.

[20] Sebastian Deorowicz, Improvements to Burrows-Wheeler compression algorithm. Software Practice and Experience, vol. 30, no. 13, 2000.

[21] Sebastian Deorowicz, Second step algorithms in the Burrows-Wheeler compression algorithm. Software Practice and Experience, vol. 32, no. 2, 2002.

[22] Peter Fenwick and Mark Titchener and Michelle Lorenz, Burrows Wheeler - Alternatives to Move to Front. url citeseer.ist.psu.edu/563772.html

[23] Stinson, D, Cryptography: Theory and Practice. CRC Press, 1995. 3.2.1.

[24] P.A. Wayner, A Redundancy Reducing Cipher. Cryptologia 107-112, 1988.

[25] Ruy Luiz Milidiú, Claudio Gomes de Mello, and José Rodrigues Fernandes, A Huffman-based text encryption algorithm. SSI - Computer Security Symposium, 2000.

[26] Ruy Luiz Milidiú, Claudio Gomes de Mello, and José Rodrigues Fernandes, Adding Security to compressed information retrieval systems. SPIRE - String Processing and Information Retrieval, 2001.

[27] Ruy Luiz Milidiú, Claudio Gomes de Mello, and José Rodrigues Fernandes, Substituição Homofônica Rápida via Códigos de Huffman Canônicos. WSeg - Workshop on Computer Systems Security, 2001.

[28] Fernandes, José Rodrigues, Algoritmo Homofônico Canônico para Cifragem e Compressão Simultâneas, Dissertação de Mestrado, Puc-Rio, 2001.

[29] Claudio Gomes de Mello, Ruy Luiz Milidiú and C.J.P. Lucena, Introducing Security into Prefix-free Encoding Schemes. Second IEEE International Security In Storage Workshop, 2003.

[30] Chung-E Wang, PhD. Cryptography in Data Compression, SAM, 2003.

[31] Milidiu, R.L., Mello, C.G. Ramdomized Huffman codes, PUCRioInf. MCC49/04 December, 2004. 1, 3.4.1, 5.3

[32] Milidiu, R.L., Mello, C.G. Adding Security to Prefix Codes, PUCRioInf. MCC50/04 December, 2004

[33] Milidiu, R.L., Mello, C.G. A provably secure crypto-compression algorithm, CIBSI 05 - 3o Congreso Iberoamericano de Seguridad Informática, Chile, 2005.

[34] Milidiu, R.L., Mello, C.G. Crypto-compression prefix coding, DCC 2006 - Data Compression Conference, USA, 2006.

[35] Maj. Mello, C., Codificação Livre de Contexto pra Cripto-Compressão, Tese de Doutorado, PUC-RIO, 2006.

[36] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., Introduction to Algorithms, The MIT (The Massachusetts Institute of Technology) Press, 2nd Edition, 2001.

[37] Hyungjin Kim, Jiangtao Wen, and John D. Villasenor, Secure Arithmetic Coding. IEEE Transactions on Signal Processing, 2007.

[38] Wireless lan medium access control (MAC) and physical layer (PHY) specifications. IEEE Standard 802.11, 1999 Edition. L. M. S. C. of the IEEE Computer Society.

[39] Fluhrer, S., Mantin, I. e Shamir, A., Weaknesses in the Key Scheduling Algorithm of RC4, Lecture Notes in Computer Science, Springerlink, vol. 2259, p. 1-24, 2001

[40] Mantin, I. e Shamir, A., A practical attack on broadcast RC4, In FSE: Fast Software Encryption, 2001.

[41] Golic, J, Linear statistical weakness of alleged RC4 keystream generator, In EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT, 1997.

[42] Joan Daemen, Lars Knudsen, Vincent Rijmen, The Block Cipher Square, Fast Software Encryption, Volume 1267 in Lecture Notes in Computer Science (E. Biham, ed.), pp. 149-165. Springer-Verlag, 1997.

[43] Joan Daemen and Vincent Rijmen, The Design of Rijndael: AES - The Advanced Encryption Standard, Springer-Verlag, 2002. ISBN 3-540-42580-2.

[44] Rivest, R.L., Mohtashemi, M., Gillman, D.W. On Breaking a Huffman Code, IEEE Transactions on Information Theory, vol. 42, no. 3, 1996.

[45] FIPS 140-2. http://csrc.nist.gov/cryptval/140-2.htm

[46] ANNOUNCING DEVELOPMENT OF A FEDERAL INFORMATION PROCESSING STANDARD FOR ADVANCED ENCRYPTION STANDARD/ http://csrc.nist.gov/archive/aes/index.html

[47] Announcing the Standard for DATA ENCRYPTION STANDARD (DES). http://www.itl.nist.gov/fipspubs/fip46-2.htm

[48] FIPS 197 - Annoucing AES, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[49] RFC3174 - SHA1. http://www.faqs.org/rfcs/rfc3174.html

[50] The Canterbury Corpus http://corpus.canterbury.ac.nz/descriptions/