5
Algoritmos Cripto-Compressores

5.1
Introdução

É denominado neste trabalho o termo algoritmo cripto-compressor para definir algoritmos que implementem as funções de cifragem e compressão de dados simultânemente.

Para isso, foram analisados alguns algoritmos de compressão de dados como os códigos de Huffman (22), a transformada de Burrows-Wheeler (2), códigos aritméticos (1), entre outros, e então, foram implementadas variantes destes algoritmos incluindo no seu processamento alguma técnica para garantir o sigilo dos dados codificados.

Neste trabalho o algoritmo em foco são as variantes do algoritmo de Huffman (22), mais especificamente o algoritmo de Huffman Canônico. Análises e implementações semelhantes podem ser aplicadas para outros algoritmos.

Foram mantidas as funcionalidades típicas dos sistemas de recuperação de informação, como indexação e busca, de tal forma que estas possam ser executadas sobre o texto codificado, ou seja, comprimido e cifrado.

O objetivo deste trabalho não é conseguir sigilo absoluto como seria necessário em aplicações críticas (ex: bancárias, militares, etc.), e sim apenas tornar a criptoanálise tão difícil, de tal forma que o custo da quebra do código seja tão alta que inviabilize o processo para informações não-críticas (ex: aplicações em CD, IRS, sites de busca, armazenamento pessoal, etc.).

5.2
Trabalhos Correlatos

Dado um arquivo codificado com Huffman, um tipo de ataque é a busca exaustiva como descrita por Klein et al. (8) em seu trabalho sobre o armazenamento seguro de texto em CD. Dada uma string S composta dos caracteres $\{c_1c_2c_3 \ldots c_m\}$, Huffman gera uma string de bits $\{b_1b_2b_3 \ldots b_n\}$ referentes aos códigos $\{d_1d_2d_3 \ldots d_m\}$, onde d_i=código de Huffman de c_i. Então, uma busca exaustiva consiste em particionar $\{b_1b_2b_3 \ldots b_n\}$ em m blocos não-vazios, o que significa testar uma combinação de (n-1)!/(n-1-m)!m! casos, que

têm de ser verificados quanto às seguintes condições:
a) os códigos $\{d_1, d_2, d_3, \dots\}$ distintos são códigos livres de prefixo;
b) a codificação é consistente, ou seja, se c_i=c_j então tem-se d_i=d_j.
O tamanho exponencial de m e o custo $O(n^m)$ geralmente invalidam a busca exaustiva.

Em (9), Fraenkel and Klein demonstram que a complexidade da quebra de textos codificados por códigos de prefixo é um problema NP-Completo.

Rivest et al. (28) fizeram a criptoanálise de uma mensagem codificada com Huffman na qual o criptoanalista não conhece o modelo (alfabeto de símbolos). 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.

Uma árvore de Huffman é uma árvore ótima, 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 (33) 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.

Embora recentes publicações de métodos para compressão de dados deixem a impressão de que Huffman estaria obsoleto e superado pelas novas técnicas, ele ainda continua sendo muito útil principalmente para o armazenamento de grandes massas de textos estáticos, enquanto os novos métodos adaptativos, como o de Ziv and Lempel (utilizado nas ferramentas compress, pkzip, arj, etc.) e os Códigos Aritméticos (1), são preferíveis em algumas aplicações de tempo real e para comunicação de dados.

Huffman possui uma série de vantagens, destacam-se: simplicidade; rapidez; auto-sincronização após erros na transmissão; e na busca, para encontrar uma string S em um texto comprimido T, normalmente é necessário primeiro se descomprimir T e então procurar a string, ao invés disso, como Huffman codifica cada símbolo sempre da mesma maneira, basta comprimir a string S e então procurá-la em T.

5.3 Algoritmos Propostos

Neste trabalho foram propostos 4 algoritmos que são descritos nos próximos capítulos.

ADDNULLS Inserção Seletiva de Nulos (13, 14), apresentado no capítulo 6. Este

algoritmo usa a técnica da 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. É mostrado que as perdas nas taxas de compressão são relativamente pequenas.

HHC Huffman Homofônico-Canônico (15), apresentado no capítulo 7. Este algoritmo cria uma nova árvore homofônica baseada na árvore de Huffman canônica original para o texto de entrada. Os resultados dos experimentos são mostrados.

RHUFF Huffman Randomizado (18), apresentado no capítulo 8. 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).

HSPC2 Códigos de Prefixo baseados em Substituição Homofônica com 2 homofônicos (17, 19, 20, 21), apresentado no capítulo 9. 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. É mostrado que a quebra do HSPC2 é um problema NP-Completo.