Adding Security to Prefix Codes

Ruy Luiz Milidiu
e-mail: milidiu@inf.puc-rio.br

Claudio Gomes de Mello
e-mail: cgmello@inf.puc-rio.br

PUC-RioInf.MCC50/04 December, 2004

Abstract : Data compression and ciphering are essential features when digital data is stored or trasmitted over insecure channels. Prefix codes are widely used to obtain high performance data compression algorithms. Given any prefix code for the symbols of a plaintext, we propose to add security using a multiple substitution function and a key. We show that breaking the code when we are given the ciphertext, dictionary, frequencies and code lengths is a NP-Complete problem.

Keywords : Security, compression, prefix codes, homophonic substitution.

Resumo : Compressão de dados e cifragem são funcionalidades essenciais quando dados digitais são armazenados ou transmitidos através de canais inseguros. Códigos de prefixo são largamente utilizados para se obter algoritmos de compressão de dados de alto desempenho. Dado um código de prefixo para os símbolos de um texto, propomos adicionar segurança utilizando uma função de substituição múltipla e uma chave. Demostramos que a quebra do código quando são dados o texto cifrado, o dicionário, as frequências e os tamanhos dos códigos é um problema NP-Completo.

Palavras – chave : Segurança, compressão, códigos de prefixo, substituição homofônica.

1 Introduction

Data compression and ciphering are essential features when digital data is stored or trasmitted over insecure channels. Usually, we apply two serial operations: first, data compression to save disk space and to reduce transmission costs, and second, data ciphering to provide confidenciality. This solution works fine to most applications, but we have to execute two expensive operations, and if we want to access data, we must first decipher and then decompress the block of information.

In this work we propose to add some additional strategies to prefix data compression algorithms so that we can achieve both compressed and ciphered data with the use of a single algorithm, improving computacional efficiency. Moreover, Information Retrieval (IR) features such as indexing and searching [14] can be mantained with this approach, which is not true with the serial solution.

Each cryptographic algorithm offer a different degree of security. If the cost required to break the algorithm, that is, recovering the code and/or the key, is greater than the value of the ciphered data, than we are "probably safe" [18]. Our main goal is not to reach absolute secrecy as might be needed to some critical applications such as the military, Internet Banking or e-Commerce. We want only to make cryptoanalysis hard enough so that the price of breaking the code is too high if compared to the value of the information been kept secret (CD-ROM applications, IR systems, data searching web sites, personal storage, etc.).

We examine the security impact of adding some strategies to prefix codes motivated by previous empirical findings. We extend previous results [10, 8, 9] on practical implementation of data ciphering-compressing algorithms where we used Canonical Huffman coding, dyadic distributions and some additional strategies in order to secure the ciphertext against cryptanalysis. We propose a scheme that adds security into the compression process by using a homophonic substitution algorithm and a key: the *HSPC2 - Homophonic Substitution Prefix Codes with 2 homophones*. We show that the use of homophonic substitution increases the security of the ciphertext. We assume that the adversary is given the ciphertext, dictionary, frequencies and code lengths. His goal is to break the key used during the encoding process. We show that it is a NP-Complete problem.

In the encoding process, the HSPC2 function appends a one bit suffix to some codes. A secret key and a homophonic rate parameters control this appending. According to the values of these two parameters, the algorithm chooses which instances of the symbol are going to receive the one bit suffix. We can also have different ciphertexts to the same plaintext and key due to this homophonic substitution approach. This work is related to the SPC scheme analised by Fraenkel and Klein [5], but here we use a novel approach. Fraenkel and Klein use the strategy of having a fixed suffix of variable size

per symbol in the plaintext. In their work, the size and the presence (or not) of the suffix is the secret. The key size is equal to the number of symbols in the plaintext. On the other hand, we use homophonic substitution, so that we have the ocurrence of a one bit suffix defined by a key and a rate parameter. If the symbol key is not set the symbol does not have the one bit suffix, otherwise the symbol can have or not the one bit suffix and its presence is defined by the homophonic rate. The key size is equal to the number of symbols in the dictionary.

In Section III we describe the proposed the homophonic substitution function. In section IV, we evaluate the algorithm's security strength. In Section V, the compression loss is analysed and the data expansion due to the homophonic substitution approach is shown to be asymptotically smaller than 5% per character, under usual parsing and coding assumptions. Finally in section VI, we present our conclusions and some guidelines to future work.

2 Related Work

To add ciphering to Huffman codes, Wayner [19] proposed a simple scheme for assigning a secret key to a Huffman tree. In his scheme, he obtains a new optimal tree by operating an exclusive-or (XOR) between each branch of the tree with a secret key. A simple version of this approach is to assign one bit of a secret key for each level of the tree, operating a XOR for each Huffman code with the corresponding position of the key. A limitation of this approach is that the key size can be too small, say $O(\log m)$, where m is the number of codewords. Furthermore, several schemes have been proposed to reduce the maximum codeword length in order to increase decoding speed [20, 13, 11].

Klein et al. [4] analysed the cryptographic aspects of Huffman codes used to encode a large natural language on CD-ROM. And in [5], Fraenkel and Klein show that this problem is NP-complete for several variants of the encoding process.

Rivest et al. [16] cryptoanalysed a Huffman encoded text assuming that the cryptoanalyst does not know the codebook. According to them, cryptoanalysis in this situation is surprisingly difficult and even impossible in some cases due to the ambiguity of the resulting encoded data.

Multiple, or also called homophonic, substitution is an old technique that transforms a plaintext sequence of symbols into a more random one. Each symbol has multiple homophones that can be choosen to represent it. This technique avoids statistical attacks to prefix-codes [2, 7].

In [10, 8, 9] we use the homophonic approach to enhance a canonical Huffman coding. Data compression rates in the range of 40% to 50% are shown. Furthermore, observed coding and decoding times were very close to the standard Canonical Huffman codes. Moreover, experiments show

that when using dyadic distributions we can even increase the encoding speed generating only a small loss in compression rates. A dyadic probability distribution is a distribution in which each probability is a negative integer power of 2. As an example, the following distribution is dyadic: $(2^{-2}, 2^{-2}, 2^{-2}, 2^{-3}, 2^{-4}, 2^{-4})$. It is shown that dyadic distributions increase the secrecy of the ciphertext since prefix coding of a dyadic distribution source leads to a random stream of bits [9].

3 The Homophonic Substitution Function

The security issues that arise when using data compression schemes have been examined by cryptographers over the years. It is well known that compressing data is not secure enough against simple analysis such as statistical attacks. To protect data against statistical analysis, Shannon [17] suggested that the language redundancy should be reduced before encryption and obscured with confusion and difusion.

In this work we use data compression algorithms to remove redundancy and homophonic substitution to obscure redundancy over confusion. Diffusion can be also added by some initial permutation or symmetric cryptographic algorithm (we do not use diffusion in this work since it can be included before or after our encoding as in [8]). We propose a security enhancement to prefix codes that we call *HSPC2 - Homophonic Substitution Prefix Codes with 2 homophones*. HSPC2 can be added to any prefix code such as static Huffman or Canonical Huffman [12] codes.

Now we describe how we construct the HSPC2, our homophonic substitution function.

3.1 Defining which symbols are to be coded with homophones

Consider the plaintext T given by `bcaabbbcc`. For this plaintext let us assume the prefix codebook defined by table 1. To define which symbols have homophones we use a secret key K. K is a binary vector: if $k_i = 1$ then symbol i is coded with a homophonic substitution function, otherwise, if $k_i = 0$, a simple substitution is used.

For example, assume that we have a dictionary of 3 distinct symbols $\{a, b, c\}$ and we choose symbols a and b to be assigned homophonic codes, that is, the secret key is $K = (1, 1, 0)$. The extended homophonic code is given by table 2.

The introduction of extra homophonic prefix codes results in compression loss. In order to control this side effect, we introduce a homophonic rate parameter. In this example, let us set it to $\frac{1}{3}$, meaning that one out of three occurrences of the symbol is to be coded with the homophonic codes. To

Table 1: Original codebook.

Symbol	Code
a	00
b	1
c	01

Table 2: Homophonic codebook.

Symbol	Homophones
a	00, 000, 001
b	1, 10, 11
c	01

obtain an integer occurrence rate for the corresponding homophonic symbols we adopt the rounding up value as shown on table 4.

The homophonic rate of a given symbol i is defined by

$$h_i = k_i.\lceil \frac{q}{m}.f_i \rceil$$

where k_i is the secret bit key of symbol i with $k_i \in \{0, 1\}$ and $i \in [0, n]$, q is an integer value with $q \in (0, m]$, m is the size of the plaintext and f_i is the frequency of symbol i with $f_i \in \aleph$.

Table 3 summarizes the encoding scheme where $q = 1$ and $m = 3$. Applying these codes to the plaintext `bcaabbbcc` obtaining the ciphertext `101000011110101`01.

If $k_i = 0$ than the homophonic rate $h_i = 0$, that means that simple substituion is used. Otherwise, h_i is a fraction $\frac{q}{m}$ of the frequency f_i of symbol i. So, q can be used to control the proportion between symbols that have homophones and symbols that do not. Key K and homophonic rate h_i defines 3 possible encoding for a symbol, the HSPC2 encoding function.

Table 3: Homophonic transformation.

Symbol	Homophonic rate h_i	Homophones
a	1	00, 000, 001
b	2	1, 10, 11
c	0	01

Table 4: Homophonic rates.

Symbol	Key k_i	Frequency f_i	Homophonic rate h_i
a	1	2	1
b	1	4	2
c	0	3	0

3.2 HSPC2 encoding

Here we formalize the HSPC2 encoding function, that introduces security into prefix encoding schemes. HSPC2 obscures redundancy, by applying homophonic substitution (confusion), over removed redundancy (data compression).

Let us be given a prefix code $C = (c_1, ..., c_n)$ for a dictionary $\Sigma = (\sigma_1, ..., \sigma_n)$ of n distinct symbols, a plaintext $T = t_1...t_m$ with $t_i \in \Sigma$, a binary vector $K = (k_1, ..., k_n)$ as the secret key with $k_i \in \{0, 1\}$, the frequency f_i of each distinct symbol in Σ, a real number $q \in (0, m]$ with $m = f_1 + ... + f_n$, an index generation function D and a binary vector $B = (b_1, ..., b_n)$ with $b_i \in \{0, 1\}$. HSPC2 generates a ciphertext S, such that $S = H_K(T) = H_{k_{t_1}}(t_1)...H_{t_m}(t_m)$, where H is the transformation given by

$$H_i = \begin{cases} c_i & \text{if } k_i = 0 \\ c_i & \text{if } k_i = 1 \text{ at rate } f_i - h_i \\ c_i.b_i & \text{if } k_i = 1 \text{ at rate } h_i \end{cases}$$

where $H_i \equiv H_{k_{t_i}}(t_i)$ and $c_i \equiv C(t_i)$.

Observe that C must be a consistent prefix code, that is, the set $c_1, ..., c_n$ must be a prefix set. Hence, no codeword c_i is a prefix of another c_j, where $i \neq j$, and if $c_i = c_j$ then $i = j$.

When a symbol has its key set ($k_i = 1$), the number of instances of this symbol that must be encoded with homophone $c_i.b_i$, is equal to h_i. In example 1, symbol a has one occurrence encoded as $00.b_i$, and one occurrence encoded as 00. We set the ceiling for h_i since $\frac{q}{m}.f_i$ is not always an integer. We set the ceiling operator to reduce the occurrence of zero rates. The floor operator generates more zeros than ones, and so, fewer homophones $c_i.b_i$, and we think that this would weaken the HSPC2 scheme. In the next section we show that either the use of floor or ceiling operator results in valid choices to our purposes.

If $k_i = 1$, HSPC2 uses either c_i or $c_i.b_i$ to encode symbol σ_i at rate h_i.

3.3 Defining which symbols were coded with homophones

The problem that arises at decoding is that when we are parsing code c_i we do not know if the next bit in the encoded stream is b_i or is part of the next symbol. So, we must use some index generation function D to solve this identification problem. Function D chooses which symbols are to be encoded as $c_i.b_i$ when $k_i = 1$. It generates indexes that are used by HSPC2 at encoding time to output codes. For example, from tables 4 and 3 we have that symbol a is encoded using code 000 or 001 only once ($r_a = 1$), and that symbol b is encoded 10 or 11 twice ($r_b = 2$). Then, the possible index vectors for the first symbol are $\{(0,1),(1,0)\}$, and for the second we have $\{(0,0,1,1),(0,1,0,1),(0,1,1,0),(1,0,1,0),(1,0,0,1),(1,1,$
$0,0)\}$. For example, vector $(0,0,1,1)$ means that symbol b has to be encoded as $c_i.b_i$ by the third and fourth time it appears, the two first encoding must use code c_i. For example, `111010`, `111011`, `111111` and `111110` are valid encodings to plaintext `bbbb`.

These vectors are generated by D and are regenerated at decoding time, so these vectors do not need to be stored. They are used only for symbols with key k_i set. We have several options to implement D such as

- a hash function
- a randomized function
- some deterministic rule

Note that algorithms must deal with colisions. A function $D \equiv D(j,x)$ with index j and some input x must be defined when implementing HSCP2. Examples of possible functions D are

- $D(j,x) = (x + D(j-1,x)) \bmod f_i$
- $D(j,x) = rand(x.D(j-1,x)) \bmod f_i$
- $D(j,x) = (2.j) \bmod f_i$

We have that the result of $D(j,x)$, given some fixed input x, is the j-th index. In the examples above, f_i is the frequency of symbol i and x is some input acting as a key. x can be the key K concatened, that is, x is the decimal value of the binary $k_1k_2k_3...k_n$. For example, if we have function D defined as $D(j,x) = (x + D(j-1,x)) \bmod f_i$ with $D(-1,x) = 0$, key $K = (1,1,0)$ and so $x = 110 = 6$, then symbol a has indexes $(1,0)$ since $D(0,6) = 6 \bmod 2 = 0$ and symbol b has indexex $(1,0,1,0)$ since $D(0,6) = 6 \bmod 4 = 2$ and $D(1,6) = (6+2) \bmod 4 = 0$.

The binary suffix b_i can be arbitrarily chosen. Some simple policies to set b_i are

- at random

- alternating 0s and 1s, that is, $b_i = 0$ if i is even and $b_i = 1$ if i is odd
- based on some fixed rule, for example: $b_i = 0$ if (i mod 10) = 0, and 1 otherwise.

3.4 HSPC2 decoding

The HSPC2 decoding function uses the default prefix decoding algorithm plus the knowledge of the key K and function D to decode the extra bits inserted into the ciphertext. Given a string of bits of the ciphertext, the HSPC2 decoding function performs the following steps

```
(1) Generate decoding tables and load the
    dictionary and codebook;
(2) Generate index vectors for each symbol
    using function D;
(3) While not end of ciphertext do
    (3.1) Decode next symbol: parse the stream
          of bits until finding a valid symbol;
    (3.2) Update the symbol counter x;
    (3.3) If the key and the index are set then
          skip next (extra) bit.
```

4 Breaking the Code

Now that we have the HSPC2 encoding function defined, we make a theoretical analysis of the difficulty of decoding the ciphertext stream. To prove the algorithm security, let us start this section by introducing an example that illustrates what we call the HSPC2 problem of breaking the code.

Example 1: Suppose we are given $< \Sigma, F, R, S, D, q, m >$, *where* $\Sigma = (a, b, c)$, $F = (2, 4, 3)$, $R = (2, 1, 2)$, $S =$ `11010010011110101`, $q = 3$ *and* $m = 9$. *Are there a consistent prefix code* C, *a plaintext* T *and a key* K *such that* $H_K(T) = S$, *where* $H_K(T)$ *is the HSPC2 function ? Assume that* $D(j, x) = (x + D(j-1, x))$ *mod* f *with* $D(-1, x) = 0$ *with* $D(-1, x) = 0$ *and* $b_i = 1$ *for all* $i \in [1, n]$.

The answer in this case is yes, since the required items can be chosen as $C = (00, 1, 01)$, $T =$ *bcaabbbcc and* $K = (1, 1, 0)$. *The core of this challenge is to find a triple* $< C, T, K >$ *that satisfies the requirements. Is it a difficult problem ? If someone faces a ciphertext* S *like this one, would it be easy to derive the plaintext* T *?*

We analyse these questions in the following sections. If we can prove that it is a computacionally difficult (NP-Complete) problem, than we can use this encoding scheme to introduce security into prefix codes.

The HSPC2 problem is defined as the following decision problem.

4.1 The HSPC2 decision problem

Input:	a dictionary $\Sigma = (\sigma_1, ..., \sigma_n)$ of n symbols, a vector $F = (f_1, ..., f_n)$ of frequencies, a vector $R = (r_1, ..., r_n)$ of code lengths, a ciphertext S obtained by $S = H_K(T)$ and $q \in (0, m]$ with $m = f_1 + ... + f_n$.
Question:	are there a consistent prefix code C, a plaintext T and a key K such that $H_K(T) = S$, where $H_K(T)$ is the HSPC2 function ?

Note that only the code lengths are provided. In fact, the vector R is given since it is an intrinsic information that can be derived from Σ, F and the knowledge of the adopted prefix coding algorithm. For example, if HSPC2 uses Huffman then R is not secret because it can be derived from the knowledge of F. An interesting feature here is that Σ and F are not secrets, and hence, do not need to be protected. Moreover, the prefix code C is not completely defined by R, due to the remaining ambiguity [16]. Furthermore, choices made in the encoding process (like varying between left-order and right-order Huffman coding) also contribute to increase the security of the ciphertext

4.2 A Simpler problem

To better understand the HSPC2 decision problem and to make analysis more feasible, let us consider a simpler problem denoted by RHSPC2. RHSPC2 is obtained by adding the two following simplifications over HSPC2.

The plaintext T is given. The plaintext T we use in RHSPC2 has a fixed layout defined by a concatenation of symbols, that is, $T = (\sigma_1^{f_1}.\sigma_1^{f_2}\sigma_1^{f_3}...\sigma_n^{f_n})$. As an example, let $T = aabbbbccc = a^2b^4c^3$. As a consequence, we have that the layout of T and the vector of frequencies F defines T.

The codebook C is given. From the knowledge of the plaintext, the code lengths and frequencies, it is possible to build a valid consistent prefix code C. Hence, in RHSPC2, the codebook is given.

The RHSPC2 problem is defined as the following decision problem.

Input:	a dictionary $\Sigma = (\sigma_1, ..., \sigma_n)$ of n symbols, a vector $F = (f_1, ..., f_n)$ of frequencies, a plaintext $T = (\sigma_1^{f_1}.\sigma_1^{f_2}\sigma_1^{f_3}...\sigma_n^{f_n})$, a codebook $C = (c_1, ..., c_n)$, with code lengths $R = (r_1, ..., r_n)$, an integer L, with $L \geq f_1 r_1 + ... + f_n.r_n$ and $q \in (0, m]$ with $m = f_1 + ... + f_n$.
Question:	is there a key K such that $\mid H_K(T) \mid = L$, where $H_K(T)$ is the HSPC2 function ?

Only the size of $H_K(T)$ is important. Note that more than one codebook can result with the same size as $H_K(T)$, that is, we can have codebooks C_1 and C_2, with the same code lengths, both resulting in $\mid H_K(T) \mid = L$. This ambiguity is pointed out by [16]. In RHSPC2 problem, one valid codebook is given, but others can exist.

4.3 Evaluating the algorithm security

According do NIST Security Evaluation Criteria used in AES [15], we show the HSPC2 security with the following algorithm properties.

4.3.1 Relative security to others

This work is mainly similar to [5], where Klein et al. show that the problem of analysing Huffman codes used to encode large natural language is NP-complete for several variants of the encoding process. In [5], Klein et al. analyse the SPC scheme where they use the strategy of having a fixed suffix of variable size per symbol in the plaintext. In their work, the size and the presence (or not) of the suffix is the secret. The key size is equal to the number of symbols in the plaintext.

On this work we use homophonic substitution, so that we have the ocurrence of a one bit suffix defined by a key and a rate parameter. If the symbol key is not set the symbol does not have the one bit suffix, otherwise the symbol can have or not the one bit suffix and its presence is defined by the homophonic rate. The key size is equal to the number of distinct symbols in the plaintext.

4.3.2 Mathematical basis - HSPC2 is NP-Complete

Now, we show that breaking the key used in the ciphertext $S = H_K(T)$ is a NP-Complete problem. We use a reduction from SUBSET-SUM, a well-known [1] NP-Complete problem.

The SUBSET-SUM (SUS) problem is defined as

Input:	a vector $A = (\alpha_1, ..., \alpha_n)$, $\alpha_i \in \aleph$, and a goal g, $g \in \aleph$, with $g \leq (\alpha_1 + ... + \alpha_n)$.
Question:	is there a binary vector $K = (k_1, ..., k_n)$ such that $(\alpha_1.k_1 + ... + \alpha_n.k_n) = g$?

Theorem 1 (SUS $\propto$ RHSPC2)

Proof 1 *We want to prove that RHSPC2 is NP-Hard, that is, there exists a reduction algorithm Λ from SUS to RHSPC2. The following three conditions must hold.*

(i) Λ builds an instance of RHSPC2 from SUS. *Suppose that the SUS problem is defined by $A = (\alpha_1, ..., \alpha_n)$ and goal g. Let us build the RHSPC2 problem from the SUS problem: generate a dictionary $\Sigma = (\sigma_1, ..., \sigma_n)$ with n symbols. Choose an integer $q \in (0, m]$ with $m = f_1 + ... + f_n$. The frequency of each symbol is defined by $f_i = \lfloor \frac{m}{q}.\alpha_i \rfloor$. The plaintext is defined as $T = (\sigma_1^{f_1}.\sigma_1^{f_2}\sigma_1^{f_3}...\sigma_n^{f_n})$. Generate a codebook applying the prefix coding algorithm defined for HSPC2. L is defined by $L =\mid S_0 \mid +g$, hence $L \in \aleph$.*

(ii) Λ is polynomial. *The tasks of generating the dictionary, choosing q, calculating f_i, defining T and obtaining a codebook (using Huffman [3] codes, for instance) are all polynomial operations. Hence, the total complexity is polynomial.*

(iii) Λ is correct. *Let K be a given binary vector. We must show that K* is a solution to SUS defined by $< A, g >$ if and only if K is a solution to RHSPC2 defined by $< \Sigma, F, T, C, R, q, m, L >$ with $L = (\mid S_0 \mid +g)$. *First, suppose that K is a solution to RHSPC2, so we have $\mid H_K(T) \mid = L$. Let us calculate $\mid H_K(T) \mid$*

$$
\begin{aligned}
\mid H_K(T) \mid &= \mid S_0 \mid + \sum_{i=1}^{n} \lceil \frac{q}{m}.f_i \rceil.k_i = \\
&= \mid S_0 \mid + \sum_{i=1}^{n} \lceil \frac{q}{m}.\lfloor \frac{m}{q}.\alpha_i \rfloor \rceil.k_i = \\
&= \mid S_0 \mid + \sum_{i=1}^{n} \lceil \frac{q}{m}.((\frac{m}{q}.\alpha_i) - \epsilon) \rceil.k_i = \\
&= \mid S_0 \mid + \sum_{i=1}^{n} \lceil \alpha_i - \frac{\epsilon}{\frac{m}{q}} \rceil.k_i
\end{aligned}
$$

$$= \mid S_0 \mid + \sum_{i=1}^{n} \lceil \alpha_i - \delta \rceil . k_i$$
$$= \mid S_0 \mid + \sum_{i=1}^{n} \alpha_i . k_i$$

with $0 \leq \epsilon < 1$, $0 \leq \delta \leq \epsilon < 1$, $\frac{m}{q} \in \Re$, $\frac{m}{q} \geq 1$ *and* $\alpha_i \in \aleph$. *Hence,* $\mid S_0 \mid + \sum_{i=1}^{n} \alpha_i . k_i = L = \mid S_0 \mid + g$, *then* $\sum_{i=1}^{n} \alpha_i . k_i = g$. *So* K *is a solution to the SUS problem.*

Now, suppose that the RHSPC2 problem is defined by $< \Sigma, F, T, C, R, q, m, L >$. *Then, the corresponding SUS problem is defined by* $A = (\alpha_1, ..., \alpha_n)$ *and goal* $g = L - \mid S_0 \mid$, *where* α_i *is defined by* $\alpha_i = \lceil \frac{q}{m} . f_i \rceil$. *If* K *is a solution to SUS, then* $(\alpha_1 . k_1 + ... + \alpha_n . k_n) = g$. *Hence,* $\mid H_K(T) \mid = \mid S_0 \mid + (\alpha_1 . k_1 + ... + \alpha_n . k_n) = \mid S_0 \mid + g = \mid S_0 \mid + L - \mid S_0 \mid = L$. *Hence,* K *is a solution to the RHSPC2 problem.*

From (i), (ii) and (iii), we prove that SUS $\propto$ *RHSPC2.*

In order to prove next theorem, we consider now that all plaintexts are represented in a run-length encoding scheme. For instance, plaintext $T =$ `abaabbccc` *is to be represented as* $\widetilde{T} =$ `1a1b2a2b3c`. *This modification in HSPC2 does not generate any lack of generality and the operation of changing representation is polynomial. This run-length representation reduce the size of the resulting plaintext if all symbols in* T *are grouped. Note that this run-length representation used by HSPC2 can represent any plaintext, especially the one used by RHSPC2. On the other hand, can double the size if the symbols are distributed over the plaintext. We assume that function* D *and extra bit policy are fixed for HSPC2.*

Theorem 2 (RHSPC2 $\propto$ HSPC2)

Proof 2 *Theorem 1 proves that RHSPC2 is NP-Hard. Now, we want to prove that HSPC2 is NP-Hard, that is, there exists a reduction algorithm* Λ *from RHSPC2 to HSPC2. The following three conditions must hold.*

(i) Λ builds an instance of HSPC2 from RHSPC2. *Suppose that the RHSPC2 problem is defined by dictionary* $\Sigma = (\sigma_1, ..., \sigma_n)$ *of* n *symbols, the vector* $F = (f_1, ..., f_n)$ *of frequencies, a plaintext* $T = (\sigma_1^{f_1} . \sigma_1^{f_2} \sigma_1^{f_3} ... \sigma_n^{f_n})$, *the codebook* $C = (c_1, ..., c_n)$, *with code lengths* $R = (r_1, ..., r_n)$, *an integer* L, *with* $L \geq f_1 r_1 + ... + f_n . r_n$, *and* $q \in (0, m]$, *with* $m = f_1 + ... + f_n$.

Building the HSPC2 problem from the RHSPC2 problem is immediate: consider the same dictionary, frequencies, codeword, code lengths and q. Now, generate the plaintext $\widetilde{T}$ of HSPC2 in run-length representation: $\widetilde{T} = f_1\sigma_1 f_2\sigma_2 ... f_n\sigma_n$.

(ii) Λ is polynomial. *HSPC2 uses all input vectors of RHSPC2, except the plaintext $\widetilde{T}$ that is easily derived. Hence, the total complexity is polynomial.*

(iii) Λ is correct. *Let K be a given binary vector. We must show that K* is a solution to RHSPC2 defined by $< \Sigma, F, T, C, R, q, m, L >$ with $L =| S |=| H_K(T) |$ if and only if K, C and $\widetilde{T}$ are solution to HSPC2 defined by $< \Sigma, F, R, q >$. $\widetilde{T}$ is the run-length representation of T. *First, suppose that K is a solution to RHSPC2. Hence, $L =| S |=| H_K(T) |$. Moreover, for the key K, codeword C and plaintext T, we have that $S = H_K(T)$. Since $\widetilde{T}$ and T are equivalent representations when the symbols are grouped and we have $H_K(\widetilde{T}) = H_K(T)$. Hence, K, C and $\widetilde{T}$ are solution to HSPC2.*

Now, suppose that K, C and $\widetilde{T}$ are solution to HSPC2. Hence, $S = H_K(T)$ and $L =| S |= |H_K(\widetilde{T}) |$. Moreover, note that symbols in $\widetilde{T}$ are not necessarily grouped, but we have $| H_K(\widetilde{T}) |=| H_K(T) |= L$. Hence, we do not need to group symbols in T to verify that K is a solution to RHSPC2.

From (i), (ii) and (iii), we prove that HSPC2 is NP-Hard.

Observe that in HSPC2, the cryptoanalyst do not know the text layout, so it is necessary to guess the partitions of the ciphertext S by exhaustive search consisting in dividing the stream of bits S into m non-empty codes, where $m = f_1 + ... + f_n$, is the number of symbols in T. The brute force analysis of the number of different combinations is first pointed out by [4] and has to consider an exponential number of possible partitions in S, under the following constraints: the set of the different codewords in the sequence must be non-empty and we must eliminate the sets that are not consistent prefix codes.

Moreover, we can have the ambiguity problem in finding the codebook when cryptoanalysing HSPC2. We have ambiguous codes when it is possible to decode an encoded data using two or more valid codebooks, that is, if a bit stream can be decoded using either codebook C_1 or C_2, resulting in equally valid possible plaintexts. Therefore, prefix codes like Huffman codes lead to ambiguous encoded data. Code am-

biguity is pointed out by Rivest et al. [16], which have cryptoanalysed a Huffman encoded data assuming that the cryptoanalyst does not know the codebook. According to them, cryptoanalysis in this situation is surprisingly difficult and even impossible in some cases due to the ambiguity of the resulting encoded data.

Theorem 3 (HSPC2 is NP-Complete)

Proof 3 *HSPC2 is NP-Complete if the following two conditions hold: (i) HSPC2 is in NP; (ii) HSPC2 is in NP-Hard. To prove (i), we assume that we are given a certificate C, T and K. Then, it is immediate to verify if it is true or not that $S = H_K(T)$. We get (ii) from theorem 1. Hence, HSPC2 is NP-Complete.*

4.3.3 Resistance to cryptoanalysis

Cryptanalysis is the recovery of a plaintext or the key without access to the key. Adversaries is assumed to have complete access to ciphertext in transit over communication channel or in a local storage device. An attempted cryptanalysis is an attack. Let's make some theoretical analysis of possibles attack to HSPC2. We have the following general types of possible cryptanalytic attacks [18]. Assume that, for each type of attack, ciphertexts are always encoded with HSPC2 and with the same key. The adversary's goal is to recover the plaintext of the messages, or better, deduce the key. The adversary has the ciphertext of several messages. In this case, the difficulty of decoding the output stream is shown to be a NP-Complete problem. The NP-completeness is not considered a no-doubt measure of cryptographic strength, but it provides some mathematical evidence that one possible attack might be difficult. Complexity theory provides a methodology for analysing the computational complexity of different cryptographic techniques and algorithms. All cryptographic algorithms, except one-time pads, can be broken. The strongest statements that can be made are that known cracking algorithms are of superpolynomial time complexity.

5 Estimating Ciphertext Expansion

Inserting bits in the output stream makes cryptanalysis difficult, but reduces compression effiency. So, let's estimate the expected ciphering data expansion.

In RHSPC2, the plaintext is not a secret, and its layout is given by $T = (\sigma_1^{f_1}.\sigma_1^{f_2}\sigma_1^{f_3}...\sigma_n^{f_n})$. L is the integer defined by

$$L =\mid H_K(T) \mid=\mid S \mid,$$

with

$$L \geq \sum_{i=1}^{n} r_i.f_i$$

First, let us calculate the size of $H_K(T)$, that is,

$$\mid H_K(T) \mid = \mid S \mid = \sum_{i=1}^{n} r_i.f_i.(1-k_i) +$$

$$+ \sum_{i=1}^{n} (r_i+1).\lceil \frac{q}{m}.f_i \rceil.k_i + \sum_{i=1}^{n} r_i.(f_i - \lceil \frac{q}{m}.f_i \rceil).k_i =$$

$$= \sum_{i=1}^{n} r_i.f_i + \sum_{i=1}^{n} \lceil \frac{q}{m}.f_i \rceil.k_i = \mid S_0 \mid + \sum_{i=1}^{n} \alpha_i.k_i$$

where

$$\mid S_0 \mid = \sum_{i=1}^{n} r_i.f_i,$$

and

$$\alpha_i = \lceil \frac{q}{m}.f_i \rceil, \alpha_i \in \aleph.$$

Hence, $L = \mid H_K(T) \mid = \mid S \mid = \mid S_0 \mid + \sum_{i=1}^{n} \alpha_i.f_i$. Finding the solution of the RHSPC2 problem is equivalent to finding K such that $\mid S_0 \mid + \sum_{i=1}^{n} \alpha_i.f_i = L$. $\mid S_0 \mid$ represents a ciphertext with key $K_0 = (0, ..., 0)$.

Now, we define the ciphering data expansion Ψ of S as

$$\Psi = \frac{(\mid S \mid - \mid S_0 \mid)}{\mid S_0 \mid} = \frac{\sum_{i=1}^{n} \alpha_i.f_i}{\sum_{i=1}^{n} r_i.f_i}$$

and since $\sum_{i=1}^{n} r_i.f_i \geq \sum_{i=1}^{n} f_i = m$, we have

$$\Psi \leq \frac{\sum_{i=1}^{n} \lceil \frac{q}{m}.f_i \rceil.k_i}{m} \leq \frac{\sum_{i=1}^{n} (1 + \frac{q}{m}.f_i).k_i}{m} \leq \frac{n+q}{m}$$

since $k_i \leq 1, \forall i = 1, ..., n$.

Observe that D is an increasing function of q. Hence, q can be used to control the extra bits overhead due to ciphering. The data expansion depends also on the vectors K and F. To calculate the expected data expansion let us introduce some probabilistic assumptions. First, let us assume that q, K and F are independent. This is true for RHSPC2 since given F, the key K and the number q are chosen at random and independent of F. In this case, $E[q] = \frac{m}{2}$ and $E[k_i] = p[k_i = 1] = \frac{1}{2}$.

For a plaintext we have

$$E[\Psi] \quad \leq \quad E[\frac{\sum_{i=1}^{n} (1 + \frac{q}{m}.f_i).k_i}{m}] =$$

$$
\begin{aligned}
&= E[\sum_{i=1}^{n} \frac{k_i}{m} + \sum_{i=1}^{n} \frac{\frac{q}{m}.f_i.k_i}{m}] = \\
&= \frac{E[\sum_{i=1}^{n} k_i]}{m} + E[\frac{q}{m^2}.\sum_{i=1}^{n} f_i.k_i] = \\
&= \frac{\frac{n}{2}}{m} + \frac{1}{m^2}.E[q].E[\sum_{i=1}^{n} k_i.f_i] = \\
&= \frac{n}{2.m} + \frac{1}{m^2}.\frac{m}{2}.\sum_{i=1}^{n}(E[k_i].E[f_i]) = \\
&= \frac{n}{2.m} + \frac{1}{2.m}.\sum_{i=1}^{n}(\frac{1}{2}.E[f_i]) = \\
&= \frac{n}{2.m} + \frac{1}{4.m}.\sum_{i=1}^{n} E[f_i] = \\
&= \frac{n}{2.m} + \frac{1}{4.m}.E[\sum_{i=1}^{n} f_i] = \\
&= \frac{n}{2.m} + \frac{1}{4.m}.m \approx .25
\end{aligned}
$$

since $m \gg n$ for large plaintexts.

The expected ciphering data expansion $E[\Psi]$ is asymptotically smaller than 25% bits per symbol for usual parsing and coding assumptions. Now, if the symbols are words or n-grams, the data expansion can be asymptotically bounded as $E[\Psi] \leq \frac{.25}{t}$ for t-gram parsing codes. As an example, for a 5-gram parsing, or similarly for word parsing, the bound is 5%. Moreover, since Ψ depends on q, K and F, one can decrease data expansion adopting other policies to choose q and K. For instance, for $E[k_i] = \beta$ we have $E[\Psi] = \frac{\beta}{2}$ bits per symbol, where $\beta \in [0, 1]$. The disadvantage here is a lack of secrecy, since this different probabilistic distribution for K can be used by cryptoanalysts.

6 Conclusions and future work

The issues that arise when using data compression schemes have been examined by cryptographers over the years. It is known that compressing data is not secure enough against simple analysis such as statistical attacks. In this paper, we propose a security enhancement to the encoding process by using a homophonic substitution algorithm with a key: the *HSPC2 - Homophonic Substitution Prefix Codes with 2 homophones.*

This work shows how to add a homophonic strategy to prefix data compression algorithms. Such enhancement aims to increase the security of information retrieval systems. Through the results presented is this paper, we obtain a theoretical analysis of the security feature that is added to a

modified prefix data compression code. We also provide simple guidelines to practical implementations of data ciphering-compressing algorithms using Canonical Huffman coding, dyadic distribution and other experimental strategies intended to secure the ciphertext against cryptanalisys. We plan to integrate this new strategy to a practical implementation and test its empirical performance.

One major advantage of the HSPC2 function is that information retrieval features, such as indexing and searching [14], are kept the same. The possible overhead is some data expansion due to the encryption approach, but analyses under usual assumptions shows that, for word parsing, the compression loss is asymptotically smaller than 5% per character.

Other possible strategies can be added to this scheme to achieve lower data expansion and better performance. Therefore, the theoretical and practical impact in security due to modifications in the algorithm must be analysed. For instance, if the secret key K is dependent on the plaintext T, say $k_i = \frac{1}{f_i}$, then we can have lower data expansion, but it is still an open problem the impact of this modification into its secrecy properties. Also, other related techniques can be used to optimize the overall algorithm like skeleton trees [6] and dictionary reduction schemes to deal with large scale texts [20].

Further security analysis is also needed. For example, we must analyse the problem of weak instances of SUBSET-SUM and its relation to HSPC2. Also, we must analyse other types of attacks as the known-plaintext attack (the adversary has has access to plaintexts and the encoded ciphertexts, and try to deduce the key), the chosen-plaintext attack (the adversary chooses the plaintexts that gets encoded) and the chosen-ciphetext attack (the adversary chooses the ciphertexts to be decoded).

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L. *Introduction to Algorithms*, The MIT (The Massachusetts Institute of Technology) Press, 1990.

[2] Gunter, C.G. *An Universal Algorithm for Homophonic Coding in Advances in Cryptology*, Eurocrypt-88, LNCS, vol. 330, 1988.

[3] Huffman, D. *A Method for the Construction of Maximum of Minimum Redundancy Codes*, Proc. IRE, 1098-1101, 1952.

[4] Klein, S. T., Bookstein, A., Deerwester, S. *Storing Text Retrieval Systems on CD-ROM: Compression and Encryption Considerations*, ACM Transactions on Information Systems, vol. 7, no. 3, 1989.

[5] Klein, S.T., Fraenkel, A.S. *Complexity Aspects of Guessing Prefix Codes*, Algorithmica 12 409-419, 1989.

[6] Klein, S.T. *Skeleton Trees for the Efficient Decoding of Huffman Encoded Texts*, 8th Annual Symposium on Combinatorial Pattern Matching (CPM'97), 65-75, 1997.

[7] Massey, J.L., Kuhn, Y.J.B., Jendal, H.N. *An Information-Theoretic Treatment of Homophonic Substitution*, In Advances in Cryptology Eurocrypt-89, LNCS, vol. 434, 1989.

[8] Milidiu, R.L., Mello, C.G, Fernandes J.R. *Adding security to compressed information retrieval systems*, SPIRE - String Processing and Information Retrieval, 2001.

[9] Milidiu, R.L., Mello, C.G, Fernandes J.R. *Substituio Homofnica Rpida via Cdigos de Huffman Cannicos*, Wseg - Workshop on Computer Systems Security, 2001.

[10] Milidiu, R.L., Mello, C.G., Fernandes J.R. *A Huffman-based text encryption algorithm*, SSI - Computer Security Symposium, 2000.

[11] Milidiu, R.L., Laber, E.S., Moreno, L.O., Duarte, D.C. *A Fast Decoding Method for Prefix Codes*, Proceedings of DCC, pp.438, 2003.

[12] Moffat, A., Witten, I.H., Bell T.C. *Managing Gigabytes: Compressing and Indexing Documents and Images, second edition*, Academic Press, 1999.

[13] Lidell, M., Moffat, A. *Hybrid Prefix Codes for Practical Use*, Proceedings of DCC, pp.392, 2003.

[14] Moura, E., Navarro, G., and Ziviani, N. *Indexing compressed text*, Proceedings of the 4th South American Workshop on String Processing, 1997.

[15] Nechvatal, J., Barker, E., Bassham, L. et al. *Report on the Development of the AES*, Computer Security Division, NIST IT Lab, 2000.

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

[17] Shannon, C. *Communication Theory of Secrecy Systems*, Bell Syst. Tech., vol. 28, no. 4, pp. 656-715, 1949.

[18] Schneier, B. *Applied Cryptography Second Edition: protocols, algoritms and source code in C*, John Wiley & Sons, 1996.

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

[20] Zobel, J., Williams, H.E. *Compact In-Memory Models for Compression of Large Text Databases*, SPIRE - String Processing and Information Retrieval, 1999.