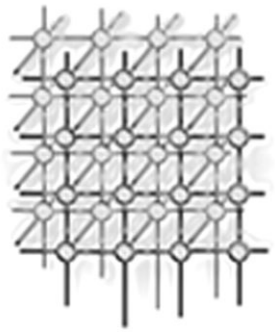
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2004; **16**:1063–1076 (DOI: 10.1002/cpe.804)



Augmenting LZ-77 with authentication and integrity assurance capabilities

Mikhail J. Atallah[1] and Stefano Lonardi[2,*,†]

[1]*CERIAS and Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, U.S.A.*
[2]*Department of Computer Science and Engineering, University of California, Riverside, CA 92521, U.S.A.*

SUMMARY

The formidable dissemination capability allowed by the current network technology makes it increasingly important to devise new methods to ensure authenticity and integrity. Nowadays it is common practice to distribute documents in compressed form. In this paper, we propose a simple variation on the classic LZ-77 algorithm that allows one to hide, within the compressed document, enough information to warrant its authenticity and integrity. The design is based on the unpredictability of a certain class of pseudo-random number generators, in such a way that the hidden data cannot be retrieved in a reasonable amount of time by an attacker (unless the secret bit-string key is known). Since it can still be decompressed by the original LZ-77 algorithm, the embedding is completely 'transparent' and backward-compatible, making it possible to deploy it without disrupting service. Experiments show that the degradation in compression due to the embedding is almost negligible. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: authentication; integrity; data compression; LZ-77; fragile watermark

1. INTRODUCTION

The growing concern about authenticity, confidentiality, and the protection of intellectual property on the Internet has recently raised the interest in information hiding. The main areas of research in information hiding are *steganography* and *watermarking*. Whereas steganography is the science of concealing the existence of secret messages within larger ones from an external observer, watermarking

*Correspondence to: Stefano Lonardi, Department of Computer Science and Engineering, University of California, Riverside, CA 92521, U.S.A.
†E-mail: stelo@cs.ucr.edu

Contract/grant sponsor: Purdue Research Foundation; contract/grant number: 690-1398-3145
Contract/grant sponsor: National Science Foundation; contract/grant numbers: EIA-9903545 and ISS-0219560
Contract/grant sponsor: Office of Naval Research; contract/grant number: N00014-02-1-0364
Contract/grant sponsor: Center for Education and Research in Information Assurance and Security

Received March 2003
Revised June 2003
Accepted June 2003

can be thought as a stronger version of steganography, since it usually requires robustness against attacks aimed at removing the watermark (for an introduction see, e.g., [1–5]).

A new type of watermark, called *fragile*, has recently been proposed for digital images [6–8]. A fragile watermark is designed to ensure that the document cannot be changed without destroying the watermark. The most immediate application is document authenticity and integrity. The watermark becomes some sort of a 'witness' to the authenticity and integrity of the document.

So far, most of the research in watermarking has been focused on multimedia (images, audio, video) and source code. Techniques that hide messages in English texts range from line-shifting, word-shifting and font encoding (see, e.g., [9–11]) to natural language processing approaches (see, e.g., [12]). Whereas the first groups of techniques are particularly sensitive to attacks by optical character recognition, the second groups subtly change the text and that may not be tolerable for some applications. In general, the problem of watermarking English text appears to be quite challenging.

The difficulty in hiding information in textual data is justified in the literature by the following consideration [1, p. 332]

> ... text is in many ways the most difficult to hide data ... due largely to the relative lack of redundant information in a text file as compared with a picture or a sound file ...

also repeated in [5, p. 36]

> ... unlike noisy data, written text contains less redundant information which could be used for secret communication ...

Any practitioner of lossless data compression knows that texts can be lossless-compressed equally or sometimes even more than an image or an audio file. This observation sparked our initial idea of hiding information within a compressed representation of text.

It is also well known that compression aids encryption by reducing the redundancy of the plain text. The lower the redundancy of the plain text being fed to an encryption algorithm, the more difficult is the cryptanalysis of that algorithm [13,14]. Compression has been proposed as a method of encryption [13,15,16], but several attacks have been studied: on Huffman encoded texts [17], dynamic on Huffman encoded texts [18], but in particular on arithmetic coded files [16,19–21].

To the best of our knowledge, the work by Cachin [22] is the only one which combines information hiding and textual data compression. He modifies Willems' algorithm [23] to illustrate a steganographic system in which he proves to achieve asymptotically perfect security.

The scheme by Ziv and Lempel [24] (LZ-77) implemented in the popular `gzip`/`zip` family of archivers has nowadays become a standard. Files are distributed over the Internet, commonly in compressed form. In Section 2 we first consider the problem of hiding a secret message M within the Lempel–Ziv compressed representation of a document. The objective is to make sure that M cannot be retrieved by the attacker in any reasonable amount of time (unless the secret bit-string key is known). The security of the system is the topic of Section 3. In Section 4 we describe how to use our findings for document authentication. Some experiments are reported in Section 5.

2. HIDING MESSAGES IN LZ-77

The typical cryptographic protocol involves two parties, traditionally named Alice and Bob. We use T to denote the document that Alice desires to send to Bob, and M the secret message. We assume T over

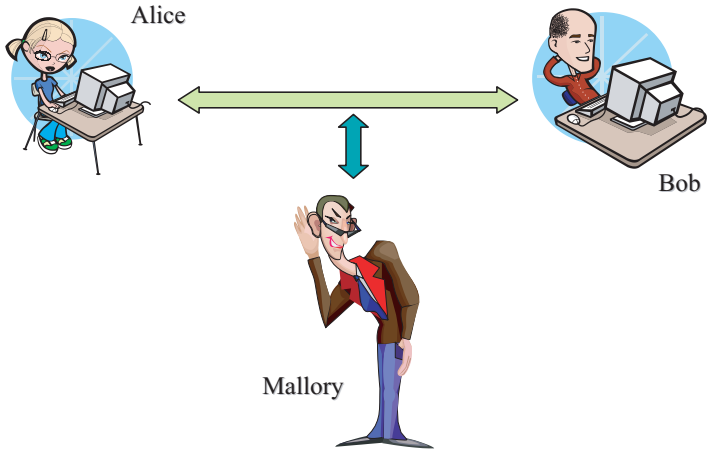


Figure 1. While Alice and Bob are exchanging some documents, Mallory is eavesdropping on the conversation and devising ways to tamper with the files.

an alphabet Σ, and M over the alphabet $\{0, 1\}$. The text T should be 'long enough' to accommodate M, as explained below.

When we decompose the text T in uvw, i.e. $T = uvw$ where u, v and w are strings over Σ, strings u, v and w are called *substrings*, u is called a *prefix* of T, and w is called a *suffix* of T. Given a string T, the number of symbols in T defines the *length* $|T|$ of T. Throughout this document, we assume $|T| = t$ and $|M| = m$.

We write $T_{[i]}$, $1 \leq i \leq t$ to indicate the ith symbol in T. We use $T_{[i,j]}$ shorthand for the substring $T_{[i]}T_{[i+1]} \ldots T_{[j]}$, where $1 \leq i \leq j \leq t$, with the convention that $T_{[i,i]} = T_{[i]}$. Substrings in the form $T_{[1,j]}$ correspond to the prefixes of T, and substrings in the form $T_{[i,t]}$ to the suffixes of T.

We denote by H a one-way cryptographic hash function (for example, MD5 [25]).

Suppose that our two friends, Alice and Bob, desire to exchange T and they want to ensure that what they receive is authentic and integral, i.e. they want to establish some proof of authorship that cannot be forged or reused. Sitting between Alice and Bob, however, there is Mallory. Mallory is eavesdropping on the conversation and devising a plan on how to tamper with T (see Figure 1).

Before starting the exchange of documents, Alice and Bob agreed upon (and memorized) a secret key k. We suppose that the key k is the *only* parameter unknown to Mallory, as the Kerckhoffs' principle dictates. The idea is that even if Mallory understood the internals of the system, he should never be able to change T and get away with it without knowing k. In fact, we want to ensure that Mallory is completely prevented from retrieving the secret message M from the compressed text.

When Alice decides to send (T, M) to Bob, she compresses the text T with a modified version of LZ-77, called LZS-77 ('S' for secret), which secretly embeds M. A description of the original LZ-77 is in order, followed by the one for LZS-77.

The LZ-77 algorithm [24] processes the data *on-line* as it is read, i.e., parses the file sequentially *left to right* and looks into the sequence of past symbols to find a match with the longest prefix of the

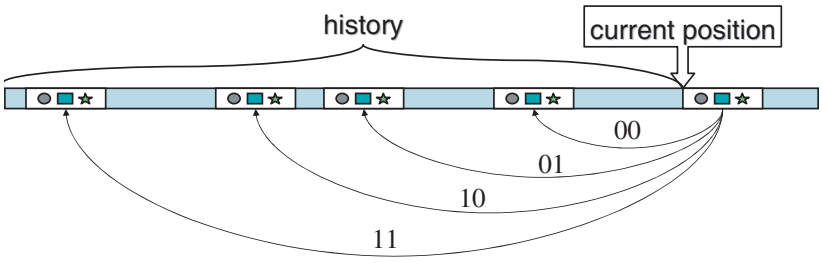


Figure 2. The multiplicity of the next phrase is four ($q = 4$). Choosing one of the four possible pointers hides two bits of the secret message.

string starting at the current position. The longest prefix is substituted with a *pointer*, which is a triple composed of *(position, length, symbol)*. Several variations on LZ-77 have been proposed (see, e.g., [26] and references therein), but the basic principle remains the same.

Let us suppose that the first $i-1$ symbols of the string T have already been parsed in $n-1$ phrases, i.e. $T_{[1,i-1]} = y_1 y_2 \dots y_{n-1}$. To identify the nth phrase, LZ-77 looks for the *longest* prefix of $T_{[i,t]}$ that matches a substring of $T_{[1,i-1]}$. If $T_{[j,j+l-1]}$, $j < i$ is the substring that matches the longest prefix, then the next phrase is $y_n = T_{[i,i+l-1]}$. The algorithm issues the pointer $(j, l, T_{[i+l]})$ and updates the current position i to $i + l + 1$. The reason we need $T_{[i+l]}$ is to be able to advance when $l = 0$, which is common in the very beginning.

In the LZS-77 algorithm, we slightly modify the LZ-77 encoding to be able to embed M. We define a position i in the text corresponding to the beginning of a new phrase to have *multiplicity* q if there exists exactly q matches for the longest prefix that starts at position i of T. The positions with multiplicity $q > 1$ are the places where we embed some bits of the secret message M. Specifically, the next $\log_2 q$ bits of M will secretly drive the selection of one particular pointer out of the q choices (see Figure 2).

Suppose again that the initial portion of T, say $T_{[1,i-1]}$, has already been parsed. Let $\{(p_0, l, T_{[i+l]}), (p_1, l, T_{[i+l]}), \dots, (p_{q-1}, l, T_{[i+l]})\}$, $q \geq 1$ be the set of feasible pointers for the longest prefix of $T_{[i,t]}$, where $l > 1$, and $1 \leq p_l \leq i$ for all $0 \leq l \leq q-1$. In particular, we consider the positions with multiplicity $q > 1$. When $q = 1$ we simply skip to the next phrase.

When $q > 1$, we first generate a random permutation of the set $S = \{0, 1, \dots, q-1\}$ as follows. We store S in a balanced data structure which supports the operation $\text{EXTRACT}(S, n)$ in time $O(\log q)$, such as a 2-3 tree (there are tree schemes that achieve $O(\log \log q)$ performance, but they are mostly of theoretical interest). The operation $\text{EXTRACT}(S, n)$ returns and simultaneously removes the nth smallest element in S. We generate a pseudo-random sequence $a_1, a_2, \dots$ using BBS [27], with seed $a_0 = H(k, i, p_0, p_1, \dots, p_{q-1})$. Then, for each $j = q-1, q-2, \dots, 1, 0$ we set $b_j = \text{EXTRACT}(S, a_j \bmod (j+1))$. It is easy to prove that $\{b_0, b_1, \dots b_{q-1}\}$ is a uniformly distributed permutation of S. We use the random permutation to re-order the pointers as $R = \{(p_{b_0}, l, T_{[i+l]}), (p_{b_1}, l, T_{[i+l]}), \dots, (p_{b_{q-1}}, l, T_{[i+l]})\}$.

We now assign a unique binary code to each pointer by building the tree of the optimal binary prefix code of a uniform distribution on q symbols. We first write the multiplicity q in binary notation as $q_{K-1} \dots q_1 q_0$, where $K = \lfloor \log_2(q) \rfloor + 1$ and $q = \sum_{j=0}^{K-1} 2^j q_j$. We build a complete binary tree B of

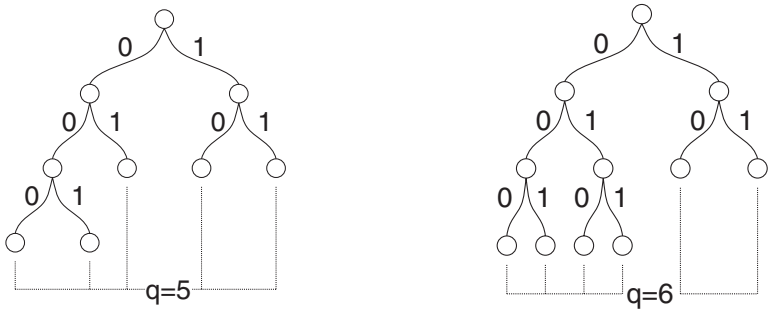


Figure 3. The trees for $q = 5$ and $q = 6$.

height K, where each node has a unique code identified by the path from the root to the node, using the convention that going to a left child corresponds to a '0' and going to a right child corresponds to a '1'. At the end of the process described next each pointer will be assigned to a node of the tree B and hence it will have a unique binary label. We initially assign the first 2^{K-1} pointers of the set R to the nodes at level $K-1$ of tree B. If there remain unlabeled pointers, that is if $q > 2^{K-1}$, we consider the $q - 2^{K-1}$ leftmost nodes at level $K-1$. For each of these, we 'move' the pointer down to its left child and we assign the next unlabeled pointer to the right child.

Figure 3 shows the two trees for the cases $q = 5$ and $q = 6$. For example, if $q = 5$ we get the code $\{000, 001, 01, 10, 11\}$ which has an average length of 2.4 bits, while $\log_2 5 = 2.3219$. If $q = 6$ we get the code $\{000, 001, 010, 011, 10, 11\}$ which has an average of 2.6666 bits, while $\log_2 6 = 2.5839$.

Finally, we use the next bits of M to choose one of the q pointers. Suppose that the first $r-1$ bits of M have already embedded. We traverse the tree B using the longest prefix of $M_{[r,m]}$ that ends up in a node marked with a pointer, say p_{b_j}. We then emit $(p_{b_j}, l, T_{[i+l]})$, we move the current position to $i + l + 1$, and we increment r by the length of the code of p_{b_j}. The complete algorithm is summarized in Figure 4.

We want to stress that these changes do not affect the internal structure of LZ-77 encoding, other than a possible re-shuffling of the pointers. A file compressed with LZS-77 can still be decompressed by a standard LZ-77 algorithm. This backward-compatibility makes LZS-77 particularly easy to deploy gradually without disrupting service. Moreover, the compression is still on-line, i.e. the file is not required to be stored entirely in primary memory.

An important parameter for any watermarking technique is the capacity of the watermarking channel. We studied the capacity of LZS-77 channel both theoretically and experimentally. From the theoretical perspective, we proved in [28] that the average multiplicity tends to a constant as the size of the text tends to infinity. This is also confirmed by our experiments in Section 5.

In practice, if Alice finds that there are not enough positions with multiplicity $q > 1$ to encode M, she simply has to append some more irrelevant data to T. *Vice versa*, if the text is longer than she needs, she can increase the security by distributing the bits of the message in a *subset* of the set of positions with multiplicity $q > 1$. Let $L = \{l_1, l_2, \ldots, l_s\}$ be the set of positions with multiplicity $q > 1$, as collected in a full scan of the LZ parsing. She can select a random permutation of a subset of L by using the

```
LZS-77_ENCODER (T, M, k)
1  let i, r, t, m, P ← 0, 0, |T|, |M|, empty string
2  while i < t do
3     let T[i,l] ← the longest prefix of T[i,t] that matches a substring in T[1,i−1]
4     let R ← {(p0, l, T[i+l]), ..., (pq−1, l, T[i+l])} be the set of feasible pointers for T[i,l]
5     if q > 1 then
6        initialize the BBS generator with seed a0 = H(k, i, p0, p1, ..., pq−1)
7        for j ← 1, ..., q − 1 do
8           let aj ← BBS(aj−1)
9        let S ← {0, 1, ..., q − 1}
10       for j ← q − 1, ..., 1, 0 do
11          let bj ← EXTRACT(S, aj mod (j + 1))
12       let B ← tree for the optimal binary prefix code of a uniform distribution on q symbols
13       let pbj ← pointer stored in a leaf of B at the end of a path which begins at the root
         of B and spells out the longest prefix of M[r,m]
14       append (pbj, l, T[i+l]) to P
15       let r ← r + ⌊log2 q⌋
16    else
17       append (pq−1, l, T[i+l]) to P
18    let i ← i + l
19    return P
```

Figure 4. The algorithm for the encoder. T is the text, M is the secret message, k is the secret key and P is the sequence of pointers.

same algorithm described above, using $H(k, l_1, l_2, \ldots, l_s)$ as a seed. The message is embedded only in the subset obtained by the random sequence. For all the positions with multiplicity $q > 1$ not used, she randomly chooses one of the possible pointers. Note however, that now the algorithm is not on-line anymore.

To decrypt the message, Bob runs his LZS-77 decompressor. Bob decodes the file by expanding, one after another, the pointers with their respective substring. However, at each expansion Bob also checks whether there are other possible pointers that could have been used to encode the current phrase, thereby identifying positions with multiplicity $q > 1$. In this case, some of the bits of the message could have been encoded. Bob builds the permutation $\{b_0, b_1, \ldots b_{q-1}\}$ and the binary tree of height $\lfloor \log_2(q) \rfloor + 1$ in the same way Alice did. The path from the root to the pointer chosen by Alice reveals the next bits of the message. The algorithm is summarized in Figure 5.

3. SECURITY

Other than tampering with the document, Mallory may try to retrieve the secret message, the key, or both. We show that if the adversary could determine some bits of the secret message then he would be


```
LZS-77_DECODER (P, k)
1  let D, M, i ← empty string, empty string, 0
2  for each (p, l, c) ∈ P do
3     if q > 1 then
4        let R ← {p_0, ..., p_{q-1}} be the set of occurrences of D_{[p,p+l-1]} in D_{[1,i]}
5        initialize the BBS generator with seed a_0 = H(k, i, p_0, p_1, ..., p_{q-1})
6        for j ← 1, ..., q − 1 do
7           let a_j ← BBS(a_{j-1})
8        let S ← {0, 1, ..., q − 1}
9        for j ← q − 1, ..., 1, 0 do
10          let b_j ← EXTRACT(S, a_j mod (j + 1))
11       let B ← tree for the optimal binary prefix code of a uniform distribution on q symbols
12       let h the index such that p_h = p
13       let p_{b_j} ← pointer stored in a leaf of B at the end of a path which begins at the root
         of B and spells out h
14       append the bits of b_i to M
15    append D_{[p,p+l-1]}c to D
16    let i ← i + l + 1
17 return (D, M)
```

Figure 5. The algorithm for the decoder. P is the sequence of pointers, k is the secret key, D is the decompressed text and M is the watermark

able to break a crypto-secure pseudo-random generator (e.g. BBS [27]), which is extremely unlikely (hence it is just as unlikely that the adversary can get the secret message bits).

Suppose that the adversary knows an algorithm $\mathcal{A}$ to retrieve the watermarks from the LZS-77 compressed text. We now describe how to design a method that correctly guesses the next bit of a BBS generator using $\mathcal{A}$ as a subroutine. We set T=`ababab`, and compress it with LZS-77, starting from the last copy of `ab`. The multiplicity of the pointer is two: we have one copy of `ab` at position 1 and one copy at position 3.

We initialize the seed $a_0 = H(k, 5, 1, 3)$, and $S = \{0, 1\}$. We run BBS to get the next random number a_1 and we set $b_1 = \text{EXTRACT}(S, a_1 \bmod 2)$ and $b_0 = \text{EXTRACT}(S, 1)$. Since $\mathcal{A}$ is supposedly able to retrieve the watermark, it would also be able to obtain b_1. The latter is equivalent to the ability to guess the next bit for a cryptographically secure pseudo-random generator, which cannot be done in a reasonable amount of computing time.

The security of the key k is based primarily on the one-way hash-function H. We recall that we use the key only to compute the seed $a_0 = H(k, i, p_0, p_1, \ldots, p_q)$. However, as the example above illustrates, a_0 is not directly used in the construction of the permutation. The element b_0 is always the last element left in S after all the other $q - 1$ elements have been randomly chosen. Even in the very unlikely scenario in which the adversary would be able to obtain a_0 from $b_0, b_1, b_2, \ldots, b_{q-1}$, he should still have the hard task of inverting the one-way function H.

4. AUTHENTICITY AND INTEGRITY

Fragile watermarks are an alternative to public-key cryptography (PKC)-based digital signatures [29] to achieve authentication. In order to be effective, the watermark has to meet the following specifications:

- the watermarked text is (semantically) identical to the original text;
- unless someone knows the secret key, it is very hard to add a watermark to a text that would prove the authenticity of the document;
- the presence of the watermark would hold up in court (i.e. the probability of a false positive is extremely small);
- the watermark cannot be read without the knowledge of the key;
- the watermarked text and the secret key are sufficient to obtain the original text, and read the watermark (i.e. it is not necessary to have the original text);
- the security of the watermark is based solely on the key, and not on the secrecy of the method;
- there is resistance to collusion by two people having a different watermarked version of the same text.

Our solution to the problem of ensuring authenticity and integrity is to use LZS-77 to hide as part of the watermark the *digest* of the document T using a one-way cryptographic hash function H (for example, MD5 [25] which produces a 128-bit digest). It is also good practice to add a certified time-stamp to the watermark to increase the resilience to certain types of attacks (see, e.g., [30]).

The watermark has to contain the digest of the text T, because otherwise Mallory could easily change the text and reuse the watermark for T in the different text. For example, a simple way to do it would be to re-map the symbols of the alphabet of T to a completely different alphabet. This would preserve the structure of the parsing and therefore the watermark would still be valid.

The scheme has some minor advantages over the PKC-based digital signatures. First, we are not sending any additional data other than the compressed text itself. As long as Bob keeps the text received from Alice on his storage in compressed form, the authentication is inseparably bound to the content and the text remains protected against tampering. This feature clearly simplifies the logistics of file manipulation. In contrast, the digital signature is naturally a distinct entity that risks being separated from the file it was supposed to protect.

A second advantage is that our technique is much more general and it allows us to embed *any* secret message. For example, one could think of some sort of self-embedding of the text in order to give Bob the opportunity to know *where* the text has been tampered with. Although it is not clear how to achieve this for texts, the idea has been explored for digital images [31].

The third and final advantage is that a casual observer would hardly imagine that a standard LZ-77 compressed file which (1) does not contain any suspicious-looking data and (2) can be decompressed by any common LZ-77 implementation, is actually protected against tamper-proofing. This could be used as a 'bait' in situations where we want to test the sophistication of the attacker.

The non-casual attacker has a way of detecting the presence of the watermark. Mallory can decompress the text, and recompress it again with the standard LZ-77. By comparing Alice's compressed text with his own, he may discover the re-shuffling of the pointers. He may realize that something unusual is going on, but our design will prevent him from recovering the content or reproducing the watermark for a different text.

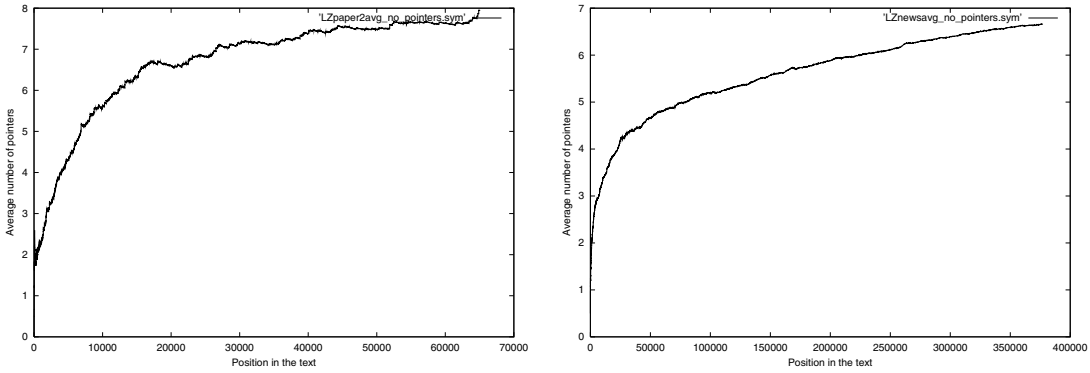


Figure 6. The average value of the pointer multiplicity q for increasing portions of `paper2` (left) and `news` (right) of the Calgary corpus.

For the reason mentioned above, this scheme cannot be used for robust watermarking and/or steganography. It appears impossible to achieve robust watermarking and/or steganography on LZ-77 compressed files. Once that Mallory has figured out that Alice is messing with the pointers, he just needs to recompress the text with the standard LZ-77 and send that instead of the compressed file received from Alice. Mallory can get rid of the message without even bothering to study the internals of LZS-77.

5. EXPERIMENTAL RESULTS

First, we illustrate that in practice there are plenty of positions with multiplicity $q > 1$ in structured texts, like English documents or software source code. We ran a few experiments on some files of the Calgary corpus, which is the standard benchmark in data compression. As a comparison, we also included a non-structured document, called `mito`, which contains the mitochondrial DNA of the yeast.

We instrumented an implementation of LZ-77 based on suffix trees [32], and we kept track of the multiplicity q for each phrase of the LZ-77 parsing, when the length of the phrase is greater than 2. The average value of q is shown in Figure 6, for increasing lengths of the prefixes. Note that for both graphs, the average for q appears to converge asymptotically to some constant.

We also measured the length of the LZ-77 phrase on the first 10 000 symbols of `paper2` (see Figure 7), and the value $\log_2 q$, which corresponds to the number of bits we could potentially embed (see Figure 8). As the figure clearly demonstrates, the number of bits that we can embed in the text T grows linearly with $|T|$. In particular, we want to attract the attention of the reader to Table I where we report how many bytes one should compress to be able to encode 128, 256, and 1024 bits.

In our opinion it is truly remarkable that one can easily store 256 bits of a secret message in less than a page of text (which is about 2000 characters). Only for non-structured texts, such as DNA, we need a longer text. In any case, about 1000 characters are enough to store an MD5 hash digest.

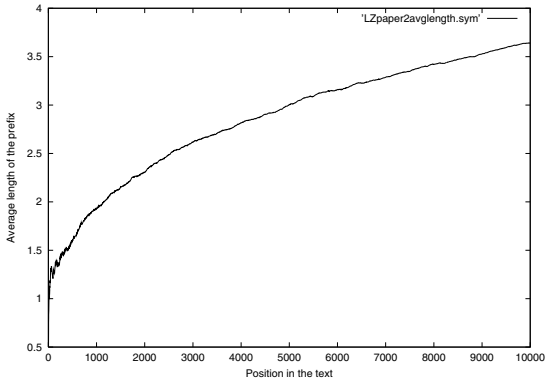


Figure 7. The average length of the LZ-77 phrase on the first 10 000 chars of `paper2`.

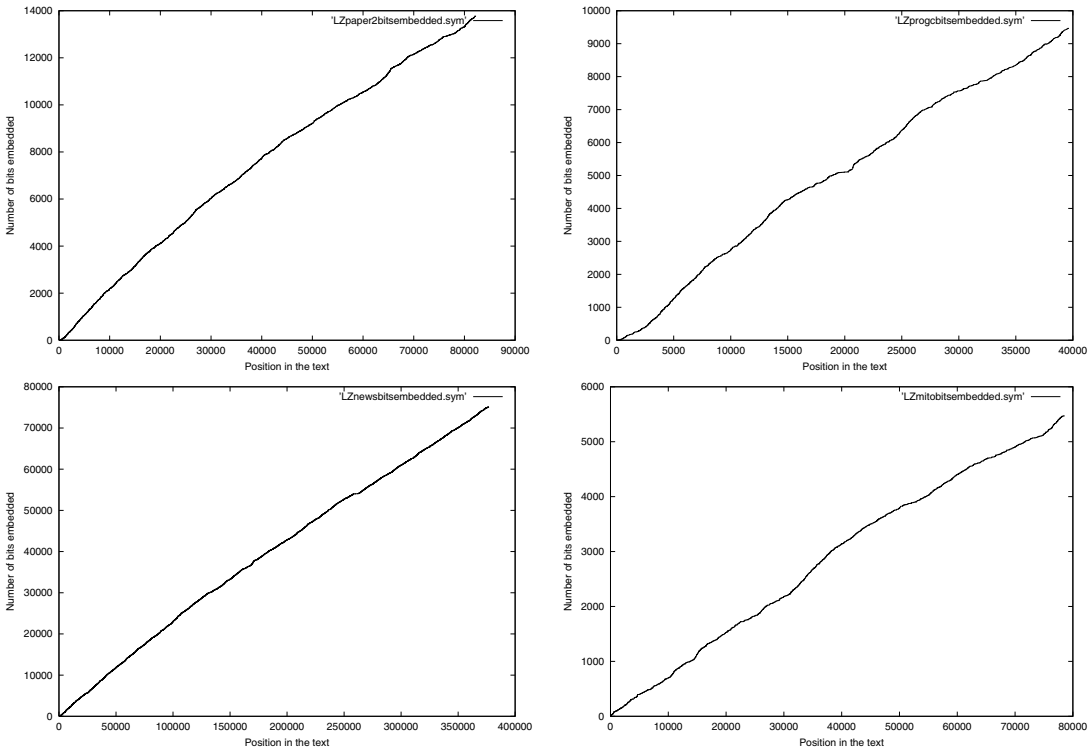


Figure 8. The number of bits embedded in `paper2` (top-left), `progc` (top-right), `news` (lower-left) and `mito` (lower-right) of the Calgary corpus (for larger and larger prefixes of the files).

Table I. The minimum length of the prefixes of texts `paper2`, `progc`, `news` and `mito` necessary to embed a given number of bits.

No. of bits embedded	Length of the prefix of `paper2`	Length of the prefix of `progc`	Length of the prefix of `news`	Length of the prefix of `mito`
128	1149	863	1115	1488
256	1692	1729	1825	3078
1024	4778	4401	5195	14 310

Table II. The compression of '`gzip -3`' versus '`gzipS -3`' for the files of the Calgary corpus; the last column shows the bits embedded by `gzipS`.

File size	`gzip`	`gzipS`	File	Bytes embedded
111 261	39 473	39 511	`bib`	1721
768 771	333 776	336 256	`book1`	14 524
610 856	228 321	228 242	`book2`	10 361
102 400	69 478	71 168	`geo`	4101
377 109	155 290	156 150	`news`	5956
21 504	10 584	10 783	`obj1`	353
246 814	89 467	89 757	`obj2`	3628
53 161	20 110	20 204	`paper1`	937
82 199	32 529	32 507	`paper2`	1551
46 526	19 450	19 567	`paper3`	893
13 286	5 853	5 898	`paper4`	249
11 954	5 252	5 294	`paper5`	210
38 105	14 433	14 506	`paper6`	738
513 216	62 357	61 259	`pic`	3025
39 611	14 510	14 660	`progc`	736
71 646	18 310	18 407	`progl`	1106
49 379	12 532	12 572	`progp`	741
93 695	22 178	22 098	`trans`	1201

Next, we modified the code of `gzip-1.2.4` to evaluate the impact of our method on compression performance. `gzip` is an optimized implementation of the *sliding window* variant of LZ-77. `gzip` is slightly different from the formal description of LZ-77 given in the previous section. `gzip` does not issue pointers in the entire history of past symbols, but only in a fixed-size window preceding the current position. This implies that the 'position' field of the pointers is a fixed size binary number (for example, 15 bits for the typical window of 32 KB). For the 'length' field, `gzip` employs eight bits which correspond to strings from 3 to 258 symbols. Strings smaller than three characters are encoded as literals.

In the presence of multiple choices `gzip` always chooses the most 'recent' occurrence of the longest prefix. The documentation explains that

> ... the hash chains are searched starting with the most recent strings, to favor small distances and thus take advantage of the Huffman encoding ...

In fact, the stream of LZ-77 pointers is encoded with Huffman. Always choosing the most recent occurrence has the effect of producing frequent short displacements that get shorter representations in the Huffman tree.

The compression performance of the `gzipS` (which implements LZS-77) with respect to the original `gzip` is illustrated in Table II on the Calgary corpus dataset. As one can expect, the embedding of the secret message degrades the compression performance. The degradation, however, is quite limited, in the order of 1–2% on average for the files in the Calgary corpus. It is worth noting that the difference in length between the files produced by `gzip` and `gzipS` is *smaller* than the numbers of bytes embedded.

This suggests a variation on LZ-77, which could also improve compression by embedding a portion of the text that one wants to compress in the selection of the pointers (instead of using the rule of always choosing the most recent occurrence).

6. CONCLUSIONS

We have shown how a simple modification to the original LZ-77 algorithm could 'upgrade' your favorite archiver with some quite powerful authentication and integrity detection capabilities. The basic idea is to drive the selection of the pointers according to the bits of the fragile watermark. The security of the system is based on the unpredictability of a certain class of pseudo-random generators. The results show that the degradation in compression performance is marginal, and surprisingly it turns out to be smaller than the number of bits of the embedded message.

Despite the popularity gained by LZ-77 because of the family of compressors `zip`/`gzip` and the image format `PNG`, several other textual compression methods are widely used. For example, LZ-78 [33] (and its variant LZW [34]) are used in `compress` and `GIF`. The LZ-78 scheme appears less prone to the sort of treatment we covered here. The only 'arbitrary choice' seems to be the initial assignment of codes to the symbol of the alphabet, which could hide a secret message. The size of the message, however, would be limited by the cardinality of the alphabet.

Finally, there is an interesting connection between our approach and the error-resilient textual compression by Storer and Reif [35]. To achieve resilience to transmission error, they make sure that each time a new phrase wa is added to the dictionary, $w \in \Sigma^*$, $a \in \Sigma$, the multiplicity of the pointers to w is at least b, where b is a predetermined constant that depends on the number and the type of errors allowed. In this way, if less than half of the b copies get corrupted, one can still decode wa by a voting process and therefore the error will not be propagated. If we had to hide information in such a modified version of LZ, we would be able to store at least $\log_2 b$ bits at *each* phrase.

ACKNOWLEDGEMENTS

The authors would like to thank W. Szpankowski for helpful discussions on the asymptotic behavior of LZ-77. The anonymous referees' comments helped to improve the paper.

Portions of this work were supported by Purdue Research Foundation Grant 690-1398-3145, by Grants EIA-9903545 and ISS-0219560 from the National Science Foundation, Contract N00014-02-1-0364 from the Office of Naval Research, and by sponsors of the Center for Education and Research in Information Assurance and Security.

REFERENCES

1. Bender W, Gruhl D, Morimoto N, Lu A. Techniques for data hiding. *IBM Systems Journal* 1996; **35**(3, 4):313–336.
2. Johnson NF, Jajodia S. Exploring steganography: Seeing the unseen. *IEEE Computer* 1998; **31**(2):26–34.
3. Petitcolas FAP, Anderson RJ, Kuhn MG. Information hiding—a survey. *Proceedings IEEE* 1999; **87**(7):1062–1078.
4. Wolfgang R, Podilchuk CI, Delp E. Perceptual watermarks for digital images and video. *Proceedings IEEE* 1999; **87**(7):1108–1126.
5. Katzenbeisser S, Petitcolas F. *Information Hiding: Techniques for Steganography and Digital Watermarking* (*Computer Security Series*). Artech House: Norwood, MA, 2000.
6. Walton S. Image authentication for a slippery new age. *Dr. Dobbs' Journal* 1995; 18–26.
7. bkcit Fridrich J. Image watermarking for tamper detection. *Proceedings IEEE International Conference on Image Processing*, vol. 2. IEEE Computer Society Press: Los Alamitos, CA, 1998; 404–408.
8. Lin E, Delp E. A review of fragile image watermarks. *Proceedings of the Multimedia and Security Workshop (ACM Multimedia '99)*. ACM Press: New York, 1999; 25–29.
9. Brassil JT, Low S, Maxemchuk N, O'Gorman L. Electronic marking and identification techniques to discourage document copying. *IEEE Journal on Selected Areas in Communications* 1995; **13**(8):1495–1504.
10. Low S, Maxemchuk N, Brassil J, O'Gorman L. Document marking and identification using both line and word shifting. *Proceedings IEEE INFOCOM*. IEEE Computer Society Press: Los Alamitos, CA, 1995; 853–860.
11. Brassil J, Low S, Maxemchuk N. Copyright protection for the electronic distribution of text documents. *Proceedings IEEE* 1999; **87**(7):1181–1196.
12. Atallah MJ, Raskin V, Crogan M, Hempelmann C, Kerschbaum F, Mohamed D, Naik S. Natural language watermarking: Design, analysis, and a proof-of-concept implementation. *Proceedings of the International Workshop on Information Hiding* (*Lecture Notes in Computer Science*, vol. 2137). Springer: Berlin, 2001; 185–199.
13. Boyd C. Enhancing secrecy by data compression: Theoretical and practical aspects. *Proceedings of Advances in Cryptology (EUROCRYPT '91)* (*Lecture Notes in Computer Science*, vol. 547), Davies DW (ed.). Springer: Berlin, 1991; 266–280.
14. Witten I, Cleary J. On the privacy afforded by adaptive text compression. *Computers and Security* 1988; **7**(4):397–407.
15. Jones DW. Application of splay trees to data compression. *Communications of the ACM* 1988; **31**(8):996–1007.
16. Liu X, Farrell PG, Boyd CA. *Resisting the Bergen–Hogan attack on adaptive arithmetic coding* (*Lecture Notes in Computer Science*, vol. 1355). Springer: Berlin, 1997; 199–208.
17. Mohtashemi M. On the cryptanalysis of Huffman codes. *Master's Thesis*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, May 1992. Also published as *Technical Report MIT/LCS/TR-617*.
18. Jones DW. Patching splay encryption to weaken chosen plaintext attacks. http://www.cs.uiowa.edu/∼jones/compress/ [2001].
19. Cleary JG, Irvine S, Rinsma-Melchert I. On the insecurity of arithmetic coding. *Computers and Security* 1995; **14**:167–180.
20. Uehara T, Safavi-Naini R. Attacking and mending arithmetic coding encryption schemes. *Proceedings Australasian Computer Science Conference*. IEEE Computer Society Press: Los Alamitos, CA, 1999; 408–419.
21. Uehara T, Safavi-Naini R. Attack on Liu/Farrell/Boyd arithmetic coding encryption scheme. *Proceedings of the Communications and Multimedia Security Joint Work Conference*. Kluwer: Boston, MA, 1999.
22. Cachin C. An information-theoretic model for steganography. *Proceedings of Workshop on Information Hiding* (*Lecture Notes in Computer Science*, vol. 1525). Springer: Berlin, 1998; 306–318.
23. Willems FMJ. Universal compression and repetition times. *IEEE Transactions on Information Theory* 1989; **1**(35):54–58.
24. Ziv J, Lempel A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 1977; **23**(3):337–343.
25. Rivest R. The MD5 message-digest algorithm. RFC 1321, MIT, RSA Data Security, April 1992.
26. Bell TC, ClearyJG, Witten IH. *Text Compression*. Prentice-Hall: Englewood Cliffs, NJ, 1990.
27. Blum L, Blum M, Shub M. A simple unpredictable pseudo-random number generator. *SIAM Journal of Computing* 1986; **15**(2):364–383.
28. Lonardi S, Szpankowski W. Joint source-channel LZ'77 coding. *IEEE Data Compression Conference (DCC)*, Snowbird, UT, March 2003, Storer JA, Cohn M (eds). IEEE Computer Society TCC, 2003; 273–283.
29. Diffie W, Hellman ME. New directions in cryptography. *IEEE Transactions on Information Theory* 1976; **22**(6):644–654.

30. Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley: New York, 1994.
31. Fridrich J, Goljan M. Protection of digital images using self embedding. *Symposium on Content Security and Data Hiding in Digital Media*. IEEE Computer Society Press: Los Alamitos, CA, 1999.
32. Rodeh M, Pratt VR, Even S. Linear algorithm for data compression via string matching. *Journal of the Association of Computing Machinery* 1981; **28**(1):16–24.
33. Ziv J, Lempel A. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory* 1978; **24**(5):530–536.
34. Welch TA. A technique for high-performance data compression. *IEEE Computer* 1984; **17**(6):8–19.
35. Storer JA, Reif JH. Error-resilient optimal data compression. *SIAM Journal of Computing* 1997; **26**(4):934–949.

