CERIAS Tech Report 2003-14

**AUTHENTICATION OF LZ-77
COMPRESSED DATA**

by Mikhail J. Atallah and Stefano Lonardi

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

Authentication of LZ-77 compressed data

Mikhail J. Atallah
CERIAS & Dept. of Computer Sciences
Purdue University
West Lafayette, IN 47907
mja@cs.purdue.edu

Stefano Lonardi
Dept. of Computer Science and Engineering
University of California
Riverside, CA 92521
stelo@cs.ucr.edu

ABSTRACT

The formidable dissemination capability allowed by the current network technology makes it increasingly important to devise new methods to ensure authenticity. 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. The design is based on the unpredictability of a certain class of pseudo-random 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". Preliminary experiments show also the degradation in compression due to the embedding is almost negligible.

Keywords

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 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., [3, 11, 15, 23, 12]).

A new type of watermark, called *fragile* has been recently proposed for digital images [20, 9, 13]. A fragile watermark is designed to ensure that the document cannot be changed without destroying the watermark. The most immediate application is document authenticity. The watermark becomes some sort of a "witness" to the authenticity 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., [5, 14, 6]) to natural language processing approaches (see, e.g., [1]). Whereas the first groups of techniques are particularly sensitive to attacks by optical character recognition, the second ones change subtly 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 ([3], page 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 [12] (page 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.

To the best of our knowledge, the work by Cachin [7] is the only one which combines information hiding and textual data compression. He modifies Willems' algorithm [22] to illustrate a steganographic system which he proves to achieve asymptotically perfect security.

The scheme by Lempel and Ziv [25] (LZ-77) implemented in the popular `gzip`/`zip` family of archivers has became nowadays a standard. Files are distributed over the Internet commonly in compressed form. In Section 2 we first consider the problem of hiding a secret mes-

sage 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 preliminary 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 call T the document that Alice wants to send to Bob, and M the secret message. We assume T over 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 i-th 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 $T_{[1,j]}$ corresponds to the prefixes of x, and substrings in the form $T_{[i,t]}$ to the suffixes of x.

We denote by H a one-way cryptographic hash function (for example, MD5 [16]).

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

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 [25] 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 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., [2] and references therein), but the basic principle remains the same.
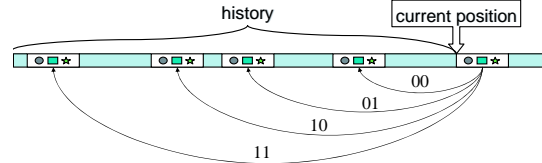

Figure 1: The multiplicity of the next phrase is four ($q = 4$). Choosing one of the four possible pointers hides two bits of the secret message

Let us suppose that the first $i - 1$ symbols of the string T have been already parsed in $n - 1$ phrases, i.e., $T_{[1,i-1]} = y_1 y_2 \ldots y_{n-1}$. To identify the n-th 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 1).

Suppose again that the initial portion of T, say $T_{[1,i-1]}$, has been already parsed. Let $\{(p_0, l, T_{[i+l]}), (p_1, l, T_{[i+l]}), \ldots, (p_{q-1}, l, T_{[i+l]})\}$, $q \geq 1$ 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 $S = \{0, 1, \ldots, q - 1\}$ 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 of mostly theoretical interest). The operation $\text{EXTRACT}(S, n)$ returns and simultaneously removes the n-th smallest element in S. We generate a pseudo-random sequence $a_1, a_2, \ldots$ using BBS [4], with seed $a_0 = H(k, i, p_0, p_1, \ldots, p_{q-1})$. Then, for each $j = q - 1, q - 2, \ldots, 1, 0$ we set $b_j = \text{EXTRACT}(S, a_j \mod (j+1))$. It is easy to prove that $\{b_0, b_1, \ldots b_{q-1}\}$ is a uniformly distributed permutation of S. We use the random permutation to re-order the pointers as $R = \{(p_{b_0}, l, T_{[i+l]}), (p_{b_1}, l, T_{[i+l]}), \ldots, (p_{b_{q-1}}, l, T_{[i+l]})\}$.

We assign now a unique binary code to each pointer by building the tree of the optimal binary prefix code of
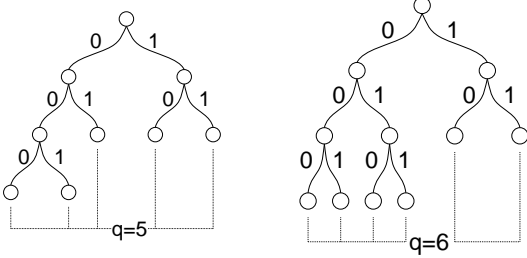
Figure 2: The trees for $q = 5$ and $q = 6$

a uniform distribution on q symbols. We first write the multiplicity q in binary notation as $q_{K-1} \ldots q_1 q_0$ 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 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 2 shows the two trees for the cases $q = 5$ and $q = 6$.

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 3.

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. 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 (see Section 5). From the theoretical perspective, we conjecture that the average multiplicity tends to a constant as the size of the text tends to infinity [19]. This conjecture is confirmed by our preliminary experiments. The details of the analysis will be reported in the full version of this paper.

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

LZS-77 (T, M, k)
1 **let** $i, r, t, m \leftarrow 0, 0, |T|, |M|$
2 **while** $i < t$ **do**
3   **let** $T_{[i,l]} \leftarrow$ the longest prefix of $T_{[i,t]}$ that matches a substring in $T_{[1,i-1]}$
4   **let** $R \leftarrow \{(p_0, l, T_{[i+l]}), \ldots, (p_{q-1}, l, T_{[i+l]})\}$ be the set of feasible pointers for $T_{[i,l]}$
5   **if** $q > 1$ **then**
6   initialize the BBS generator with seed $a_0 = H(k, i, p_0, p_1, \ldots, p_{q-1})$
7   **for** $j \leftarrow 1, \ldots, q - 1$ **do**
8   **let** $a_j \leftarrow \text{BBS}(a_{j-1})$
9   **let** $S \leftarrow \{0, 1, \ldots, q - 1\}$
10   **for** $j \leftarrow q - 1, \ldots, 1, 0$ **do**
11   **let** $b_j \leftarrow \text{EXTRACT}(S, a_j \bmod (j+1))$
12   **let** $B \leftarrow$ tree for the optimal binary prefix code of a uniform distribution on q symbols
13   **let** $p_{b_j} \leftarrow$ 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   emit $(p_{b_j}, l, T_{[i+l]})$
15   **let** $r \leftarrow r + \lfloor \log_2(q) \rfloor$
16   **else**
17   emit $(p_{q-1}, l, T_{[i+l]})$
18   **let** $i \leftarrow i + l$

Figure 3: The algorithm for the encoder. T is the text, M is the secret message and k is the secret key

$q > 1$. 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 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.

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 able to break a crypto-secure pseudo-random generator (e.g., BBS [4]), 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 $T=\texttt{ababab}$, and compress it with LZS-77, starting from the last copy of $\texttt{ab}$. The multiplicity of the pointer is two: we have one copy of $\texttt{ab}$ 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 $b_1 = \textsc{Extract}(S, a_1 \bmod 2)$ and $b_0 = \textsc{Extract}(S, 1)$. Since $\mathcal{A}$ is supposedly able to retrieve the watermark, it would be also 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 elements 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 to invert the one-way function H.

4. AUTHENTICITY AND INTEGRITY

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 [16] which produces a 128 bits 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., [18]).

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 be still valid.

The scheme has some minor advantages over the public-key cryptography-based digital signatures (PKC) [8]. 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 features clearly simplifies the logistics of file manipulation. On the contrary, the digital signature is naturally a distinct entity that risks to be 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 mes-
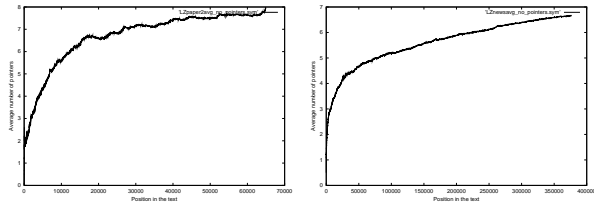

Figure 4: The average value of the pointer multiplicity q for increasing portions of `paper2` (left), and `news` (right) of the Calgary corpus

sage. 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 [10].

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 to detect 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.

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 the Alice is messing with the pointers, he just need 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 [17], 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 4, for increasing lengths of the prefixes. Note that for both graphs, the average for q appears to converge asymptotically to some constant.

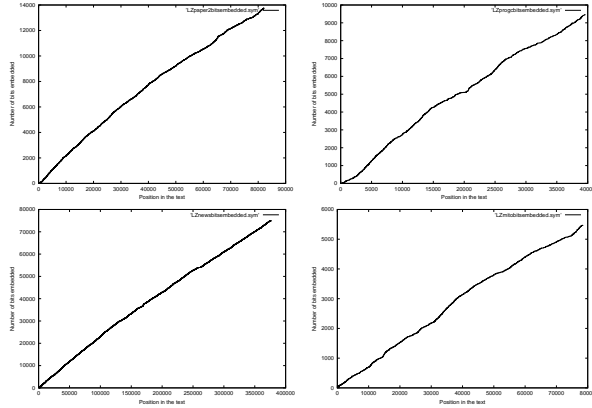We also measured $\log_2(q)$, which corresponds to the

Figure 5: 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)

# of bits embedded	length of the prefix of `paper2`	# of bits embedded	length of the prefix of `progc`
128	1,149	128	863
256	1,692	256	1,729
1,024	4,778	1,024	4,401

# of bits embedded	length of the prefix of `news`	# of bits embedded	length of the prefix of `mito`
128	1,115	128	1,488
256	1,825	256	3,078
1,024	5,195	1,024	14,310

Table 1: The minimum length of the prefixes of texts `paper2`, `progc`, `news` and `mito` necessary to embed a given number of bits

number of bits we could potentially embed. We plotted the results in Figure 5. 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 1 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 a less than a page of text (which is about 2,000 characters). Only for non-structured texts, such as DNA, we need a longer text. In any case, about 1,000 characters are enough to store an MD5 hash digest.

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, fifteen bits for the typical window of 32 Kbytes). For the "length" field, `gzip` employs eight bits which

file size	gzip	gzipS	file	bytes embedded
111,261	39,473	39,511	`bib`	1,721
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`	4,101
377,109	155,290	156,150	`news`	5,956
21,504	10,584	10,783	`obj1`	353
246,814	89,467	89,757	`obj2`	3,628
53,161	20,110	20,204	`paper1`	937
82,199	32,529	32,507	`paper2`	1,551
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`	3,025
39,611	14,510	14,660	`progc`	736
71,646	18,310	18,407	`progl`	1,106
49,379	12,532	12,572	`progp`	741
93,695	22,178	22,098	`trans`	1,201

Table 2: The compression of "gzip -3" versus "gzipS -3" for the files of the Calgary corpus; the last column shows the bits embedded by `gzipS`

correspond to strings from 3 to 258 symbols. Strings smaller than 3 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. Choosing always 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 2 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.

6. CONCLUSIONS

We have shown how a simple modification of LZ-77 could "upgrade" your favorite archiver with some quite powerful authentication 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 preliminary 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 for-

mat `PNG`, several others textual compression methods are widely used. For example, LZ-78 [26] (and its variant LZW [21]) are used in `compress` and `GIF`. The LZ-78 scheme appears less prone to the sort of treatment we did 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.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] M. J. Atallah, V. Raskin, M. Crogan, C. Hempelmann, F. Kerschbaum, D. Mohamed, and S. Naik. Natural language watermarking: Design, analysis, and a proof-of-concept implementation. In *Proc. Int. Workshop on Information Hiding*, volume 2137 of *Lecture Notes in Computer Science*, pages 185–199. Springer-Verlag, Berlin, 2001.

[2] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, 1990.

[3] W. Bender, D. Gruhl, N. Morimoto, and A. Lu. Techniques for data hiding. *IBM Systems Journal*, 35(3&4):313–336, 1996.

[4] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 15(2):364–383, May 1986.

[5] J. Brassil, S. Low, N. Maxemchuk, and L. O'Gorman. Electronic marking and identification techniques to discourage document copying. *IEEE Journal on Selected Areas in Communications*, 13(8):1495–1504, 1995.

[6] J. T. Brassil, S. Low, and N. F. Maxemchuk. Copyright protection for the electronic distribution of text documents. *Proceedings of the IEEE*, 87(7):1181–1196, 1999.

[7] C. Cachin. An information-theoretic model for steganography. In *Proc. Workshop on Information Hiding*, volume 1525 of *Lecture Notes in Computer Science*, pages 306–318. Springer-Verlag, Berlin, 1998.

[8] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[9] J. Fridrich. Image watermarking for tamper detection. In *Proceedings of the IEEE International Conference on Image Processing*, volume 2, pages 404–408, 1998.

[10] J. Fridrich and M. Goljan. Protection of digital images using self embedding. In *Symposium on Content Security and Data Hiding in Digital Media*, 1999.

[11] N. F. Johnson and S. Jajodia. Exploring steganography: Seeing the unseen. *IEEE Computer*, 31(2):26–34, 1998.

[12] S. Katzenbeisser and F. Petitcolas. *Information Hiding: Techniques for steganography and digital watermarking*. Computer Security Series, Artech House, 2000.

[13] E. Lin and E. Delp. A review of fragile image watermarks. In *Proc. of the Multimedia and Security Workshop (ACM Multimedia '99)*, pages 25–29, 1999.

[14] S. Low, N. Maxemchuk, J. Brassil, and L. O'Gorman. Document marking and identification using both line and word shifting. In *Proceedings of IEEE INFOCOM*, 1995.

[15] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn. Information hiding – a survey. *Proceedings of the IEEE*, 87(7):1062–1078, 1999.

[16] R. Rivest. The MD5 message-digest algorithm. RFC 1321, MIT, RSA Data Security, Apr. 1992.

[17] M. Rodeh, V. R. Pratt, and S. Even. Linear algorithm for data compression via string matching. *J. Assoc. Comput. Mach.*, 28(1):16–24, Jan. 1981.

[18] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, Inc., New York, NY, USA, 1994.

[19] W. Szpankowski. Personal communication, 2002.

[20] S. Walton. Image authentication for a slippery new age. *Dr. Dobbs' Journal*, pages 18–26, April 1995.

[21] T. A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, June 1984.

[22] F. M. J. Willems. Universal compression and repetition times. *IEEE Trans. on Information Theory*, 1(35):54–58, 1989.

[23] R. Wolfgang, C. I. Podilchuk, and E. Delp. Perceptual watermarks for digital images and video. *Proceedings of the IEEE*, 87(7):1108–1126, July 1999.

[24] M. M. Yeung and F. C. Mintzer. Invisible watermarking for image verification. *Journal of Electronic Imaging*, 7(3):578–591, July 1998.

[25] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, May 1977.

[26] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, Sept. 1978.