ON THE SECURITY OF A SECURE LEMPEL-ZIV-WELCH (LZW) ALGORITHM

Shujun Li[1] and Chengqing Li[2,3] and Jay C.-C. Kuo[4]

[1]Department of Computer and Information Science,
University of Konstanz, Germany
[2]College of Information Engineering,
Xiangtan University, China
[3]Department of Electronic and Information Engineering,
The Hong Kong Polytechnic University, Hong Kong SAR, China
[4]USC Viterbi School of Engineering,
Southern California University, USA

ABSTRACT

This paper re-evaluates the security of a secure Lempel-Ziv-Welch (LZW) algorithm proposed at ICME'2008. A chosen-plaintext attack is proposed to break all ciphertext indices corresponding to single-symbol dictionary entries. For short plaintexts the chosen-plaintext attack works well because string-symbol strings appear very frequently. The number of required chosen plaintexts is at the order of the alphabet size. The complexity of the chosen-plaintext attack is $O(ML)$, where M is the number of chosen plaintexts and L is the size of the ciphertext. The chosen-plaintext attack can also be generalized to chosen-ciphertext attack. In addition to the security problem, we point out that the secure LZW algorithm has a lower compression efficiency compared with the original LZW algorithm. Finally we propose several enhancements to the secure LZW algorithm under study.

Index Terms— LZW, compression, encryption, chosen-plaintext attack, chosen-ciphertext attack

1. INTRODUCTION

The idea of compressing data based on an adaptive dictionary was first proposed by Ziv and Lempel in their seminal papers published in 1977 and 1978 [1, 2]. Many variants of their original designs (called LZ77 and LZ78, respectively) have been proposed afterwards [3, 4, 5], among which LZW algorithm proposed by Welch [4] is probably the most popular one. The success of LZW was mainly due to its use in the UNIX compression tool compress and in the widely-used lossless image format GIF.

Some researchers studied how to combine dictionary coding with encryption to achieve joint encryption-compression. In [6], Xie and Kuo proposed a secure LZ78 algorithm by using multiple dictionaries and randomly selecting one secret dictionary for each encoded string. In [7], Zhou et al. proposed a secure LZW algorithm based on random dictionary insertion and permutation. Compared with naive encryption of compressed bitstream produced by a dictionary encoder, the secure dictionary encoders proposed in [6, 7] are expected to be less computationally heavy (or at least with a comparable computational complexity) because the encryption is seamlessly embedded into the dictionary encoding process. Both schemes are also designed not to compromise the coding efficiency while still maintaining a high level of security against advanced attacks especially chosen-plaintext and chosen-ciphertext attacks.

In this paper, we re-evaluate the security of the secure LZW algorithm proposed in [7] and report a chosen-plaintext attack that can break all single-symbol strings encoded-encrypted by the secure LZW algorithm. This makes it possible to partially reveal the plaintext. The chosen-plaintext attack is practical because 1) the number of required chosen plaintexts is at the order of the alphabet size, 2) the computational complexity of the attack is linearly bounded by the number of chosen plaintexts and the size of each plaintext. A chosen-ciphertext attack can be developed in a similar way, but requires more chosen ciphertexts. The chosen-plaintext attack was validated by a MATLAB implementation and experiments on textual data. To the best of our knowledge, the attacks presented in this paper are the first ones breaking the Secure LZW algorithm. Communications with the authors of [7] also confirmed the correctness of our proposed attacks.

The rest of the paper is organized as follows. In the next section we briefly describe the secure LZW algorithm under study and the security claims given in [7]. The chosen-plaintext attack is presented in Sec. 3 with experimental results. Section 4 discusses how a chosen-ciphertext attack can be developed following the same idea. Section 5 points out

that the secure LZW algorithm has a worse compression efficiency than the original LZW algorithm. In Sec. 6, we look at the question if and how the secure LZW algorithm can be enhanced. The last section concludes this paper.

2. ZHOU ET AL.'S SECURE LZW ALGORITHM

The secure LZW algorithm consists of three parts: a key scheduler, a modified secure LZW encoder and an XOR encipher. The modified LZW encoder is the core of the secure LZW algorithm and works with a dictionary of size 2^b, where b is an even number. For each encoded string S_i in an alphabet $\mathbb{A}$ whose size is n, the modified LZW encoder outputs a b-bit dictionary index X_i, which is then further encrypted by a b-bit masking key K_i to form a ciphertext index B_i.

Two secure mechanisms are involved in the modified LZW encoder: random insertion of dictionary entries and random permutation of all dictionary entries. For each new string x (including all initial entries corresponding to single-symbol strings) to be added into the dictionary, a keyed hash function is used to randomly select an unoccupied position in the dictionary. At the end of each encoding step, all dictionary entries are randomly permuted under the control of the four $(b/2)$-bit parameters c_1, c_2, r_1, r_2. The random permutation process can be described in the following steps:

- arrange all 2^b dictionary entries in a $2^{b/2} \times 2^{b/2}$ array;
- each odd column is circularly shifted by c_1 entries;
- each even column is circularly shifted by c_2 entries;
- each odd row is circularly shifted by r_1 entries;
- each even row is circularly shifted by r_2 entries.

Figure 1 shows how the key scheduler, the modified LZW encoder and the XOR encipher are connected with each other to form the whole secure LZW encipher. The random parameters c_1, c_2, r_1, r_2 and the keystream K are produced by a stream cipher (RC4) taking key as the secret key.

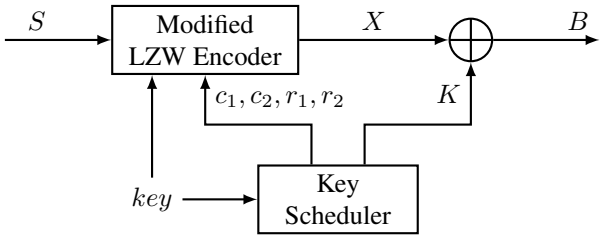

Fig. 1. The diagram of the secure LZW algorithm.

In [7], Zhou et al. analyzed the security of the secure LZW algorithm and claimed that the complexity of the ciphertext-only attack is $(2^b)!$ and the complexity of chosen-plaintext attacks is 2^{bL}, where L is the size of the ciphertext counted in b-bit integers.

3. CHOSEN-PLAINTEXT ATTACK

Zhou et al. bases their security claim against chosen-plaintext attack on the following assumptions: 1) the attacker has to exhaustively guess *each* masking key K_i (each element of the keystream K); 2) to guess K_i, all previous masking keys K_1, ..., K_{i-1} need to be guessed first. However, if we pay attention to only single-symbol entries in the dictionary, neither of the two assumptions holds anymore. This is due to the following fact about single-symbol dictionary entries: they are all inserted into the dictionary before the start of the whole encryption-compression process and updated at the end of each encoding step. In other words, all single-symbol entries are synchronized and *not* dependent on the plaintext. From this fact, we can easily derive the following theorem.

Theorem 1 *Given two different plaintexts S, S^*, if S_i and S_i^* are both single-symbol strings, then $B_i = B_i^* \Leftrightarrow S_i = S_i^*$.*

Proof: For both plaintexts, denote the dictionaries after i encoding steps (i.e., after i ciphertext indices are produced and the random permutation is done) by D_i and D_i^*. According to the description of the secure LZW algorithm, we have $B_i = D_i(S_i) \oplus K_i$ and $B_i^* = D_i^*(S_i^*) \oplus K_i$. Generally, $D_i \neq D_i^*$, however, $\forall a \in \mathbb{A}$, $D_i(a) = D_i^*(a)$ always holds because $D_i(a) = D_i^*(a) = P_{i-1}(\cdots P_1(D_0(a))\cdots)$, where P_i denotes the i-th random permutation process applied to this dictionary entry. Then, because the dictionary defines a one-to-one mapping from a string to an index and XOR is invertible, we immediately get 1) if $B_i = B_i^*$ then $S_i = S_i^*$; 2) if $S_i = S_i^*$ then $B_i = B_i^*$. This completes the proof. ■

The above theorem immediately leads to a chosen-plaintext attack on single-symbol dictionary entries: choosing a number of plaintexts to build a look-up table (LUT) between all single-symbol strings and their ciphertext indices, $\{\text{LUT}_i = \{(a, D_i(a) \oplus K_i) | a \in \mathbb{A}\}_{1 \leq i \leq L}\}$, where L is the maximal size of possible ciphertexts that we need to be handled by the attack. Once $\{\text{LUT}_i\}$ is constructed, we can easily distinguish if a ciphertext index B_i corresponds to a single-string symbol by checking if B_i can be found in LUT_i. Then the plaintext symbol corresponding to B_i in LUT_i can be set as S_i.

Note that the value of L cannot be made arbitrarily large by simply choosing sufficiently long plaintexts. If all 2-symbol strings are already in the dictionary, then no any single-symbol strings will be further coded in the ciphertext. Of course, considering the typical sizes of the alphabet and the dictionary and the fact that not all 2-symbol strings are possible in most natural language, it is either impossible (when the dictionary size is not large enough[1]) or very rare that all 2-symbol strings coded in the dictionary although it remains possible in theory.

[1]For the values used in [7], the dictionary size is $2^{12} = 4096$, less than half of the number of 2-symbol strings of printable ASCII characters.

Any␣programmer␣working␣on␣mini␣or␣microcomputers␣in␣this␣day␣and␣age␣should␣have␣at␣least␣some␣exposure␣to␣the␣concept␣of␣data␣compression.␣In␣MS-DOS␣world,␣programs␣like␣ARC,␣by␣System␣Enhancement␣Associates,␣and␣PKZIP,␣by␣PKware␣are␣ubiquitous.␣ARC␣has␣also␣been␣ported␣to␣quite␣a␣few␣other␣machines,␣running␣UNIX,␣CP/M,␣and␣so␣on.␣CP/M␣users␣have␣long␣had␣SQ␣and␣USQ␣to␣squeeze␣and␣expand␣programs.␣Unix␣users␣have␣the␣COMPRESS␣and␣COMPACT␣utilities.␣Yet␣the␣data␣compression␣techniques␣used␣in␣these␣programs␣typically␣only␣show␣up␣in␣two␣places:␣file␣transfers␣over␣phone␣lines,␣and␣archival␣storage.

a) The plaintext.

Any␣programmer␣working␣on␣m*i***c*comput*s␣*␣thi*da*and␣age␣shoul*hav*at␣leas*s**exposur*to*h**ncep*of␣*ta␣***ssi*.␣I*MS-DOS***,*****lik*ARC*b*Sy*em␣En***en*A*oci*es***PKZIP**PKwa***␣ubiqui*us**C␣**al*␣be******␣qui**␣few****ac*nes*runn**UNIX*CP/M*****n*CP/M␣us*␣*******SQ**␣USQ**squeez**␣*p*****ms*Unix␣us*␣****COMPRES***COMPACT␣*i*ties*Ye*******e****chnique*use***es****m*typicall**l**ow␣up**t**la*s:␣fi***nsf****p*n**nes***arc*val****.

b) The partially revealed plaintext when the dictionary size is 2^8.

Any␣programmer␣working␣on␣m*i***c*comput*s␣*␣thi*da*and␣age␣shoul*hav*at␣leas*s**exposur*to*h**ncep*of␣*ta␣***ssi*.␣I*MS-DOS***,*****lik*ARC*b*Sy*em␣En***en*A*oci*es***PKZIP***wa***ubiqui*us**C␣**al*␣be******␣***␣few****ac*n**runn**UNIX*CP/M*****␣**␣*****l****SQ**U***s**z***p*****m*␣U*x*s*****COMPRES****ACT*ti****Ye**********e**********ypic*l******up**t**l**:****ns*r*o**p*********iv*****.

c) The partially revealed plaintext when the dictionary size is 2^{10} and 2^{12}.

Fig. 2. The results of applying the proposed chosen-plaintext attack to an exemplar text. The white spaces are shown as "␣" to make them more visible.

While we may not make L arbitrarily large, we can choose the following n plaintexts to get a fairly large value of L: $n(n-1)+2$, which can be considered as a lower bound of possible values of L, where $A(i)$ denotes the i-th element in the alphabet $\mathbb{A}$.

- *Plaintext 1*: $A(1)$, $A(1)$, $A(3)$, $A(1)$, $\cdots$, $A(n)$, $A(1)$, $A(2)$, $A(2)$, $A(4)$, $A(2)$, $\cdots$, $A(n)$, $A(2)$, $\cdots$, $A(n-1)$, $A(n-1)$, $A(n)$, $A(n)$;
- *Plaintext 2*: $A(2)$, $A(2)$, $A(4)$, $A(2)$, $\cdots$, $A(1)$, $A(2)$, $A(3)$, $A(3)$, $A(5)$, $A(3)$, $\cdots$, $A(1)$, $A(3)$, $\cdots$, $A(n)$, $A(n)$, $A(1)$, $A(1)$;
- ...
- *Plaintext* n: $A(n)$, $A(n)$, $A(2)$, $A(n)$, $\cdots$, $A(n-1)$, $A(n)$, $A(1)$, $A(1)$, $A(3)$, $A(1)$, $\cdots$, $A(n-1)$, $A(1)$, $\cdots$, $A(n-2)$, $A(n-2)$, $A(n-1)$, $A(n-1)$.

While the chosen-plaintext attack can only break single-symbol entries, the frequent occurrence of single-symbol strings can reveal part of the plaintext, especially at the very beginning of the plaintext. If the plaintext is a text containing sensitive information, such a partial leakage may lead to devastating consequence.

The chosen-plaintext attack is very efficient in the sense that it does not involve any complicated operations rather than M encryption sessions, where M is the number of chosen plaintexts. For the above setting of chosen plaintexts, the overall computational complexity is $O(ML) = O(n^3)$. Note that the value of n is fixed for a specific application. If the plaintext is English texts composed of printable ASCII characters, then $n = 98$ if we also count tab ('`\t`'), line feed ('`\n`') and carriage return ('`\r`').

We implemented the secure LZW algorithm and the proposed chosen-plaintext attack in MATLAB and tested its real performance on a number of English texts. The MATLAB code of our implementation is available at `http://www.hooklee.com/Papers/ICME2011_SecLZW.zip`. See `readme.txt` inside the zip package on how to run a simulated attack. The results of two simulated attacks on the exemplar text (the one used in Fig. 3a of [7]) are shown in Fig. 2, where the asterisks are used to represent unrevealed characters. One can see that the meaning of the first sentence can be easily guessed due to the largely revealed words and some important keywords are leaked in the following part of the text. When the dictionary size is smaller, more words tend to be revealed because the relative frequency of single-symbol strings become higher.

Note that we can actually do more based on the partially revealed plaintext. Although none of multi-symbol strings can be directly revealed by the chosen-plaintext attack, a further post-processing can be performed to explore the redundancy in the underlying language. For instance, for the first unrevealed string in Fig. 2b, we can search all the possible multi-symbol strings that have been in the dictionary, which

include “An”, “ny”, “y ”, “ p”, “pr”, “ro”, “og”, “gr”, “ra”, “am”, “mm”, “me”, “er”, “r ”, “ w”, “wo”, “or”, “rk”, “ki”, “in”, “ng”, “g ”, “ o”, “on”, “ n”, “n ”, “ m”. Apparently, most candidates can be easily excluded and only a few need to be checked in an English dictionary for validity. This process can be at least partly automated based on some pre-defined linguistic and grammatical rules.

While the proposed attack works fairly well for short texts, it does not perform well for digital images compressed using LZW algorithm. It is because the unknown sizes of multi-symbol strings will destroy the structure of the pixels thus the leaked pixels cannot be effectively used to derive the meaning of the whole image. Therefore, the secure LZW algorithm may still be used for encrypting digital images to achieve an acceptable level of security.

Finally, it deserves noting that it is possible to generalize the chosen-plaintext attack to multi-symbol strings. Since the dictionary entry corresponding to a multi-symbol string may be added at any location i, we will need to choose more plaintexts to cover all possible insertion locations. This will requires maximal ML chosen plaintexts for each multi-symbol string. To cover all multi-symbol strings of size not greater than v, we will need $\sum_{i=2}^{v} n^i ML$ chosen plaintexts. Such a generalized attack will not work as well as the one on single-symbol entries because hash collisions will happen more frequently as the number of unoccupied dictionary entries become less. The collisions will make the selected entries of some multi-symbol strings dependent on previously coded strings, i.e., dependent on the plaintext. In this reported research, we did not go further to investigate the generalized chosen-plaintext attack because we will propose some enhancements to the original secure LZW algorithm to render the proposed chosen-plaintext attacks impossible.

4. CHOSEN-CIPHERTEXT ATTACK

A chosen-ciphertext attack can be developed based on the same idea. Different from the chosen-plaintext attack, now we turn to select different ciphertext indices to identify those corresponding to single-symbol dictionary entries. Due to the existence of invalid dictionary entries, the ciphertexts have to be chosen in an incremental way by increasing i from 1 to L. That is, we first choose 2^b ciphertexts of size 1 to identify LUT_1, then we choose 2^b more ciphertexts of size 2 to identify LUT_1, and repeat the process until we reveal all the L LUTs. In this attack, we will need L times more chosen ciphertexts than the chosen plaintexts we need in the chosen-plaintext attack. This is not a big issue since the number of chosen ciphertexts is still linearly bounded. While the chosen-ciphertext attack is less efficient than the chosen-plaintext one, it may still be useful if the attacker has access only to a joint decoder-decipher, but not to a joint encoder-encipher.

5. CODING EFFICIENCY

In Sec. 4 of [7], Zhou et al. claimed that the secure LZW algorithm has the same coding efficiency as the original LZW algorithm. This is true for the simplest fixed-width edition of LZW, but not for the variable-width edition. In the original LZW algorithm, the index of a new entry added to the dictionary is always $d + 1$, where d is the index of the last added entry (also the largest index of all valid entries). This allows coding the dictionary indices with an incremental number of bits according to the current number of valid dictionary entries. Since the LZW decoder always tries to construct the same dictionary as the encoder, both sides are synchronized in the way how they interpret each coded dictionary index.

For the exemplar text shown in Fig. 2a, a variable-width LZW encoder generates 3356 bits for both $b = 10$ and 12. However, the secure LZW encoder generates 3940 bits when $b = 10$ and 4728 bits when $b = 12$. Obviously, the loss of coding efficiency is significant in both cases. Reducing the dictionary size can help reduce the loss of coding efficiency compared with the original LZW algorithm, but it will also reduces the absolute coding efficiency and the level of security. The loss of coding efficiency drops as the size of the plaintext increases, so this problem is less serious for texts significantly longer than the dictionary size.

6. POSSIBLE ENHANCEMENTS

The insecurity against the proposed chosen-plaintext and chosen-ciphertext attacks is mainly due to the independence of the ciphertext indices B_i corresponding to single-symbol strings. As a consequence, the simplest enhancement to the secure LZW algorithm is to make the behavior of the modified LZW encoder dependent on previously coded plaintext. This can only be done at the random permutation step because the random dictionary insertion does not involve the single-symbol entries at all. The easiest way to add plaintext dependence to the random permutation process is probably to produce the four parameters r_1, r_2, c_1 and c_2 by XORing the random numbers generated by the stream cipher and the hash value of the last added dictionary entry. To prevent the generalized chosen-plaintext attack on multi-symbol strings, we suggest adding plaintext dependence to the random dictionary insertion step, too. This can be done by hashing the concatenation of the to-be-added new string and the hash value of the last added dictionary entry.

The plaintext dependence cannot have any influence on the first character coded by the secure LZW algorithm because there is no any previously coded plaintext. For the characters immediately following the first character, the plaintext dependence may be too weak to avoid potential attacks. To overcome this problem, we propose to feed the LZW encoder-encipher with a session-varying initial vector of size l, where $l > 0$ is shared with the LZW decoder-decipher so that it

knows from where the real plaintext starts. The value of l may also be encoded as part of the initial vector to avoid additional communication load.

Both enhancements increase the computational load and the second one also reduces the coding efficiency. The nature of the secure LZW algorithm also makes it difficult to overcome the problem without the loss of coding efficiency as described in the previous section. As a whole, it seems to us that a secure LZW algorithm may never outperform the naive encryption of compressed bits of a LZW encoder. It remains a question if the secure LZW algorithm can be enhanced without the above drawbacks and achieve a better overall performance than naive encryption.

7. CONCLUSION

This paper points out that a recently proposed secure LZW algorithm is not secure against a chosen-plaintext attack. All single-symbol strings encoded in the ciphertext can be revealed. The chosen-plaintext attack is practical and experimental results validated its feasibility. A chosen-ciphertext attack can also be developed based on a similar idea. Some enhancements to the secure LZW algorithm are also proposed to make it more secure against the proposed attacks.

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments that help improve the presentation of the paper. We also thank Jiantao Zhou (one of the proposers of the secure LZW algorithm in [7]) for reading our paper to confirm the correctness of the proposed attack in the paper. Shujun Li was supported by a fellowship from the Zukunftskolleg ("Future College") of the University of Konstanz, Germany, which is part of the "Excellence Initiative" Program of the DFG (German Research Foundation). The work of Chengqing Li was supported by a start-up funding from the Xiangtan University under grant No. 10QDZ39.

9. REFERENCES

[1] Jacob Ziv and Abraham Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[2] Jacob Ziv and Abraham Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[3] James A. Storer and Thomas G. Szymanski, "Data compression via textual substitution," *Journal of the ACM*, vol. 29, no. 4, pp. 928–951, 1982.

[4] Terry A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[5] Ross N. Williams, "An extremely fast ZIV-Lempel data compression algorithm," in *Proceedings of the 1991 Data Compression Conference (DCC'91)*. 1991, pp. 362–371, IEEE.

[6] Dahua Xie and C.-C. Jay Kuo, "Secure Lempel-Ziv compression with embedded encryption," in *Security, Steganography, and Watermarking of Multimedia Contents VII*. 2005, vol. 5681 of *Proceedings of the SPIE*, pp. 318–327, SPIE.

[7] Jiantao Zhou, Oscar C. Au, Xiaopeng Fan, and Peter Hon-Wah Wong, "Secure Lempel-Ziv-Welch (LZW) algorithm with random dictionary insertion and permutation," in *Proceedings of 2008 IEEE International Conference on Multimedia and Expo (ICME'2008)*. 2008, pp. 245–248, IEEE.