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

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

[1] University of Konstanz, Germany
[2] Xiangtan University, Hunan, China
[3] The Hong Kong Polytechnic University, Hong Kong SAR, China
[4] University of Southern California, USA

1 Universität Konstanz 2 3 4

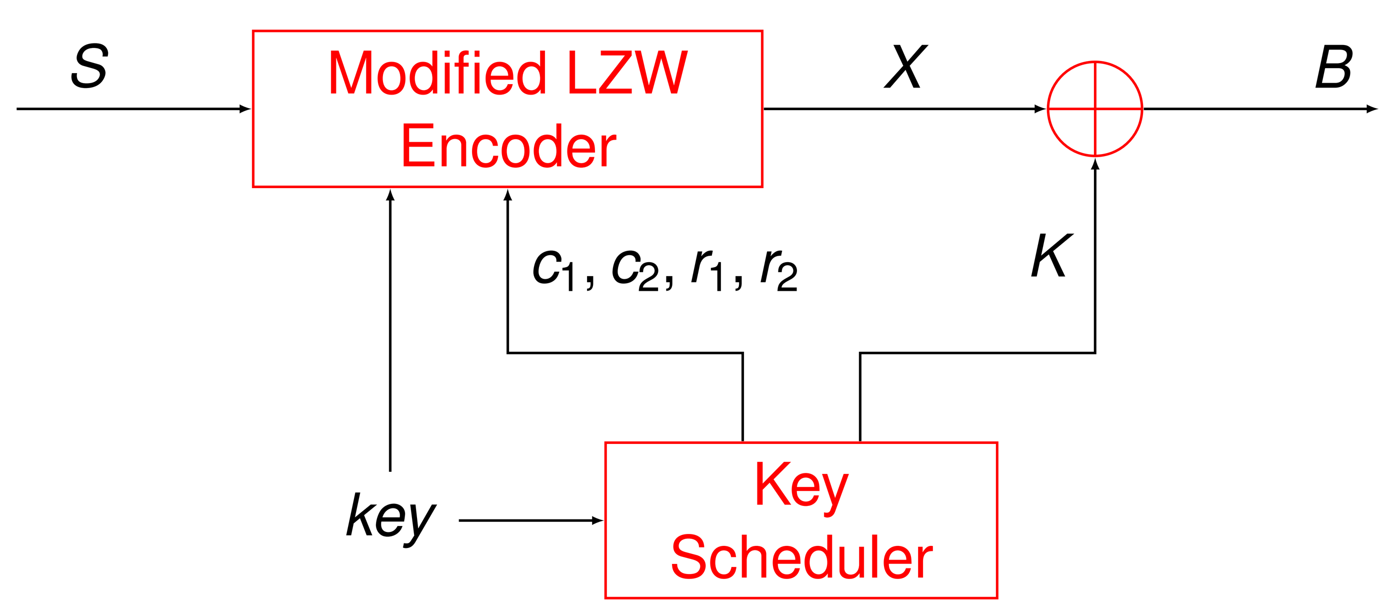
Quick Questions and Answers

1. *What is this poster about?*
   Revaluating the security of a secure LZW algorithm.
2. *Who proposed the secure LZW algorithm, when and where?*
   J. Zhou et al., "Secure Lempel-Ziv-Welch (LZW) Algorithm with Random Dictionary Insertion and Permutation", ICME 2008.
3. *What is your main finding?*
   The secure LZW algorithm is not sufficiently secure against a chosen-plaintext and a chosen-ciphertext attack.
4. *How efficient is your chosen-plaintext attack?*
   The number of required chosen plaintexts: the size of the alphabet.
   The computational complexity: $O(ML)$, where M is the number of chosen plaintexts and L is the plaintext size.
5. *And the chosen-ciphertext attack?*
   Less efficient than the chosen-plaintext attack, but still manageable.
6. *Can the security problems be overcome?*
   Yes, but at the cost of a higher computational load and/or a lower encoding efficiency.
7. *Do you have source code of your attack available somewhere?*
   `http://www.hooklee.com/Papers/ICME2011_SecLZW.zip`

Lempel-Ziv-Welch (LZW) Encoding

- LZW is a lossless coding scheme based on a dynamic dictionary.
- It is popular because of its use in the UNIX compression tool `compress` and in the lossless image format GIF.
- The encoding process works as follows:
  1. initialize the dictionary with single-symbol strings of the input alphabet;
  2. find the longest entry W in the dictionary matching the input I;
  3. add the entry index into the output and remove W from I;
  4. add a new entry Wx into the dictionary, where x is the next to-be-encoded symbol (i.e., the first symbol in I);
  5. Go back to Step 2.

Zhou et al.'s Secure LZW Algorithm



- Three security operations involved:
  - *Random insertion of dictionary entries*: the index of each dictionary entry is randomized under the control of a keyed hash function.
  - *Random permutation of dictionary entries*: the whole dictionary is organized into a square array, and then permuted columnwise and rowwise under the control of four secret parameters c_1, c_2, r_1, r_2.
  - *Output bitstream masking*: masking (encrypting) the output bitstream by XORing it with the keystream generated by a stream cipher.
- Security claims (2^b is the dictionary size and L is the plaintext size):
  - security against ciphertext-only attack: $(2^b)!$;
  - security against chosen-plaintext attack: 2^{bL}.

Security Re-Evaluation

Two problematic assumptions behind Zhou et al.'s previous security analysis:

- each masking key K_i has to be exhaustively guessed;
- K_i cannot be guessed without guessing all previous keys first.

Neither of the two assumptions holds for single-symbol entries!

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

Chosen-Plaintext Attack

- **Step 1**: Choose a number of plaintexts to build a 2-D look-up table (LUT) between all single-symbol strings S_i and their ciphertext indices B_i at each position of the plaintext.
- **Step 2**: For each ciphertext index B_i that can be found in the constructed LUT, output the corresponding single-symbol string S_i in the recovered plaintext, otherwise output "*" (an undetermined string).

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***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*****.

b) The partially revealed plaintext when the dictionary size is $2^{1}0$ and 2^{12}.

The LUT cannot be made arbitrarily large to cover plaintext of any size, but we can choose the following n plaintexts to get a fairly large LUT to break plaintext of size up to $n(n-1)+2$, where n is the alphabet size.

- *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 n*: $A(n)$, $A(n)$, $A(2)$, $A(n)$, $\cdots$, $A(n-1)$, $A(n)$, $A(1)$, $A(1)$, $A(3)$, $A(1)$, $\cdots$, $A(n-1)$, $A(1)$, $\cdots$, $A(n-2)$, $A(n-2)$, $A(n-1)$, $A(n-1)$.

Chosen-Ciphertext Attack

- Choose different ciphertext indices instead of plaintexts.
- The 2-D LUT has to be constructed in an incremental way by selecting 2^b ciphertext indices for each position of the single-symbol strings.
  $\Rightarrow$ The complexity becomes higher.

Coding Efficiency

- The secure LZW algorithm compromises the coding efficiency by disabling the possibility of using variable-width ciphertext indices.
- A comparison:
  - variable-width LZW encoder: 3356 bits,
  - Zhou et al.'s secure LZW encoder: 3940 bits when $b = 10$.

Possible Enhancements

1. Making the randomization process of dictionary entries and the random permutation process of the dictionary dependent on previously coded symbols.
2. Introducing a session-varying initial vector (IV) that obscure the first single-symbol string.

- Both enhancements increase the computational load.
- The second one reduces the coding efficiency.

Is it possible to design a secure LZW algorithm without compromising coding efficiency? – It does not seem to be likely!