# APPENDIX O
# DATA COMPRESSION USING ZIP

**William Stallings**




Supplement to
*Cryptography and Network Security, Fifth Edition*
William Stallings
Prentice Hall 2010
ISBN-10: 0136097049
http://williamstallings.com/Crypto/Crypto5e.html

PGP makes use of a compression package called ZIP, written by Jean-lup Gailly, Mark Adler, and Richard Wales. ZIP is a freeware package written in C that runs as a utility on UNIX and some other systems. ZIP is functionally equivalent to PKZIP, a widely available shareware package for Windows systems developed by PKWARE, Inc. The zip algorithm is perhaps the most commonly used cross-platform compression technique; freeware and shareware versions are available for Macintosh and other systems as well as Windows and UNIX systems.

Zip and similar algorithms stem from research by Jacob Ziv and Abraham Lempel. In 1977, they described a technique based on a sliding window buffer that holds the most recently processed text. This algorithm is generally referred to as LZ77. A version of this algorithm is used in the zip compression scheme (PKZIP, gzip, zipit, etc.).

LZ77 and its variants exploit the fact that words and phrases within a text stream (image patterns in the case of GIF) are likely to be repeated. When a repetition occurs, the repeated sequence can be replaced by a short code. The compression program scans for such repetitions and develops codes on the fly to replace the repeated sequence. Over time, codes are reused to capture new sequences. The algorithm must be defined in such a way that the decompression program is able to deduce the current mapping between codes and sequences of source data.

Before looking at the details of LZ77, let us look at a simple example. Consider the nonsense phrase

**`the brown fox jumped over the brown foxy jumping frog`**

which is 53 octets = 424 bits long. The algorithm processes this text from left to right. Initially, each character is mapped into a 9-bit pattern consisting of a binary 1 followed by the 8-bit ASCII representation of the character. As the processing proceeds, the algorithm looks for repeated sequences. When a repetition is encountered, the algorithm continues scanning until the repetition ends. In other words, each time a repetition occurs, the algorithm includes as many characters as possible. The first such sequence encountered is **`the brown fox`**. This sequence is replaced by a pointer to the prior sequence and the length of the sequence. In this case, the prior sequence of **`the brown fox`** occurs 26 character positions earlier and the length of the sequence is 13 characters. For this example, assume two options for encoding; an 8-bit pointer and a 4-bit length, or a 12-bit pointer and a 6-bit length; a 2-bit header indicates which option is

chosen, with 00 indicating the first option and 01 the second option. Thus, the second occurrence of **`the brown fox`** is encoded as $<00_b><26_d><13_d>$, or 00 00011010 1101.

The remaining parts of the compressed message are the letter **`y`**; the sequence $<00_b><27_d><5_d>$, which replaces the sequence consisting of the space character followed by **`jump`**; and the character sequence **`ing frog`**.

Figure O.1 illustrates the compression mapping. The compressed message consists of 35 9-bit characters and two codes, for a total of $35 \times 9 + 2 \times 14 = 343$ bits. This compares with 424 bits in the uncompressed message for a compression ratio of 1.24.

## O.1 COMPRESSION ALGORITHM

The compression algorithm for LZ77 and its variants makes use of two buffers. A **sliding history buffer** contains the last *N* characters of source that have been processed, and a **look-ahead buffer** contains the next *L* characters to be processed (Figure O.2a). The algorithm attempts to match two or more characters from the beginning of the look-ahead buffer to a string in the sliding history buffer. If no match is found, the first character in the look-ahead buffer is output as a 9-bit character and is also shifted into the sliding window, with the oldest character in the sliding window shifted out. If a match is found, the algorithm continues to scan for the longest match. Then the matched string is output as a triplet (indicator, pointer, length). For a *K*-character string, the *K* oldest characters in the sliding window are shifted out, and the *K* characters of the encoded string are shifted into the window.

Figure O.2b shows the operation of this scheme on our example sequence. The illustration assumes a 39-character sliding window and a 13-character look-ahead buffer. In the upper part of the example, the first 40 characters have been processed and the uncompressed version of the most recent 39 of these characters is in the sliding window. The remaining source is in the look-ahead window. The compression algorithm determines the next match, shifts 5 characters from the look-ahead buffer into the sliding window, and outputs the code for this string. The state of the buffer after these operations is shown in the lower part of the example.

While LZ77 is effective and does adapt to the nature of the current input, it has some drawbacks. The algorithm uses a finite window to look for matches in previous text. For a very

long block of text, compared to the size of the window, many potential matches are eliminated. The window size can be increased, but this imposes two penalties: (1) The processing time of the algorithm increases because it must perform a string comparison against the look-ahead buffer for every position in the sliding window, and (2) the <pointer> field must be larger to accommodate the longer jumps.

## O.2 DECOMPRESSION ALGORITHM

Decompression of LZ77-compressed text is simple. The decompression algorithm must save the last *N* characters of decompressed output. When an encoded string is encountered, the decompression algorithm uses the <pointer> and <length> fields to replace the code with the actual text string.

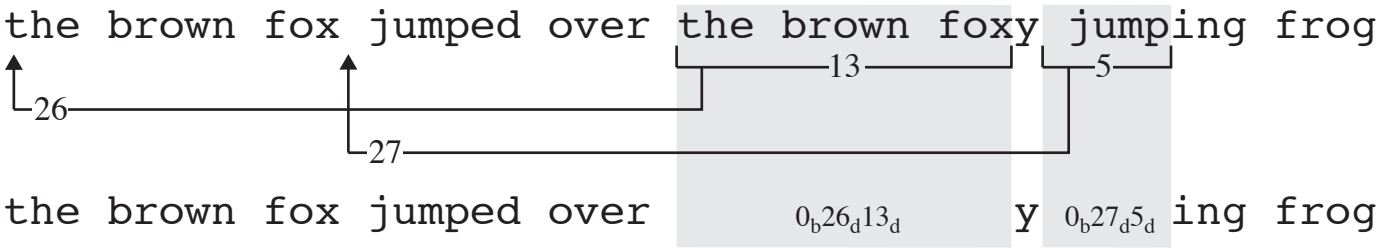
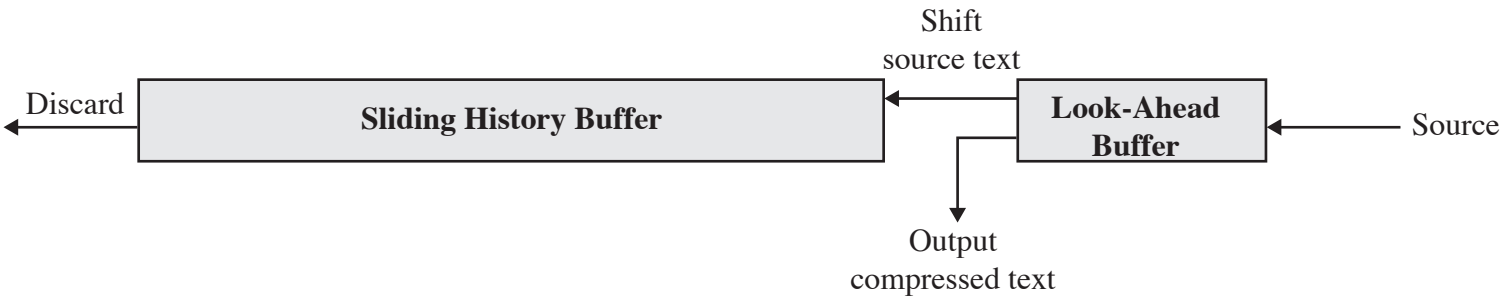
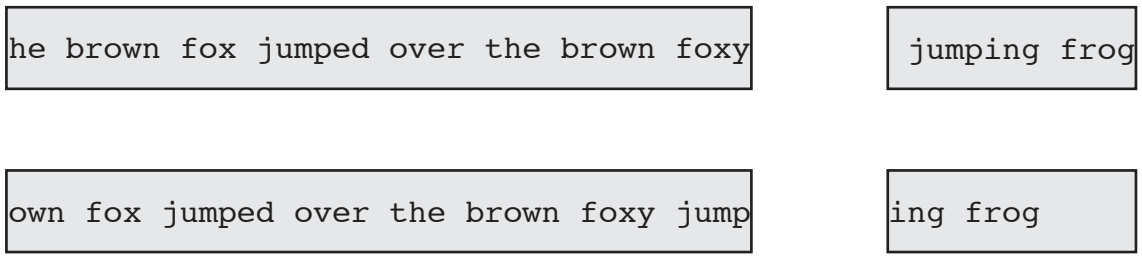



**Figure O.1 Example of LZ77 Scheme**




(a) General structure




(b) Example

**Figure O.2 LZ77 Scheme**