Document Compression and Ciphering Using Pattern Matching Technique

Sawood Alam

Department of Computer Engineering, Jamia Millia Islamia, New Delhi, India,
ibnesayeed@gmail.com

***Abstract*—This paper describes a method of document compression and ciphering using pattern matching technique. The idea behind this technique is to find out longest common pattern segments of one document in the other, and this way one document is completely referred by the other document with the help of ordered pairs of individual common pattern positions in one document and their respective lengths.**

I. INTRODUCTION

Document compression and ciphering using pattern matching technique is done with the help of two documents. One of them acts as a key and other one is the document itself which is to be compressed & ciphered. Target document is segmented as the longest common patterns among the two documents. Initial position in the key document & respective lengths of the patterns are taken as ordered pairs in a sequence in which they possess in the target document. These ordered pairs along with the key document can regenerate the target document.

This technique can be used to hide data within the data with a slight difference from traditionally used Steganography [1] techniques. Additionally it has some advantages over traditional steganography like ability to hide larger documents in smaller documents, no distortion in carrier document, document compression and better security as the key document and ordered pairs can be transmitted separately.

II. DEFINITIONS

A. Target Document

It's the document which is targeted to be compressed & ciphered using pattern matching technique.

It could be text document, image & video files, other binary files or anything else that is made of smaller element sequences.

B. Key Document

It's an arbitrary document which is being used as a reference to the target document segments. Key document is a must for ciphering & compressing the target document. The same key document is again needed while regenerating the target document from the ciphered and compressed output of the document compression and ciphering using pattern matching technique.

In accordance with the target document; key document may also be of several forms, but a key document of a particular target document must be composed of the same element grains.

In case of text type key document, it may consist of randomly jumbled characters, shuffled dictionary words or even shuffled sentences for instance.

C. Universal Document

A universal document is an arbitrary document which consists of all the element grains of a particular document type (i.e. Text document, binary document etc.) and never fails to produce a common pattern of at least single element.

D. Fine Document

It's an arbitrary, considerably large universal document that is capable of producing considerably large sequences for the entire target document of the same document type.

E. Virtually Fine Document

It's an arbitrary document that may or may not be a fine document but it can produce considerably large common patterns for a particular purpose. For example, a digest of a set of web pages of a particular web site that has high confidence value; can be a virtually fine document for that particular set of web pages.

F. Absolute Document

Absolute document of a particular document type is defined as the document with no repeated patterns & never fails to generate sequences of at least unit length. Such document is a collection of element grains taken only once for a particular document type.

G. Ideal Document

It's an extremely large universal document of a particular document type, which is (when used as key document) capable of finding whole target document as a single common pattern for any possible target document of its domain. It's an imaginary but unique document.

H. Element Grain

These are smaller pieces of a particular sequence by which documents are generated. Like in a binary sequence, 0 & 1 and in a decimal sequence digits from 0 to 9 are element grains, similarly for a text document alphabets, digits & special symbols are element grains.

I. Strange Element

A particular element of a target document that fails to get referred in a particular key document is a strange element for that key document.

J. Common Pattern

A sequence of at least one element that is common in both the documents is a common pattern. While segmenting the target document, such common sequences start after the end of the previous longest pattern sequence in the target document but the same can start from anywhere in the entire key document.

K. Ordered Pair

These are the output of pattern matching technique for compression & ciphering of a document. It's a set of 2 integers which carry start positions and lengths (optionally the end position) of the common patterns in the key document. Second element of an ordered pair may not be an integer in case of strange element. Such a case is to be handled separately.

L. Base

It's a variable which determines the start position of a pattern in the target document. It remains same until current longest pattern is being searched. Length of the current longest pattern is accumulated in the previous base to determine next value of the base.

M. Offset

It's a variable which determines the beginning position of a particular pattern in the key document. It's varying from the beginning of the key document to the end for every pattern until the longest common pattern is found.

III. COMPRESSION & CIPHERING

Compression & ciphering has time complexity in terms of the length of the key document as well as the target document. Larger the documents, more the time consumed to identify the longest common pattern. But larger key document results better compression and ciphering due to the probability of larger common sequences.

A. Objective

To find out ordered pairs of lengths & positions in the key document of longest common sequences of the entire target document in the key document; where the sequence is a prefix of the remaining target document and a subsequence of the key document.

B. Method

i. Select the target document of any type. For instance, text document. Select an appropriate key document of the same type.

ii. Use any pattern matching algorithm (e.g. Boyer-Moore [2], Knuth-Morris-Pratt [3]) to find out common sequence. For instance, take the simplest method i.e. Bruit Force Algorithm [4].

iii. Search for the very first letter of the target document in the key document. If found, note the position P of that letter in the key document from the beginning of the key document & length L equals to 1.

iv. Then check whether the second letter of the target document matches with the letter at position P+1 in the key document.

v. If it's so, then increment the value of the length by 1 & continue this process until a mismatch occurs or any one of the two documents finishes.

vi. If a mismatch occurs repeat the search for the very first letter of the target document again in the key document at position >P (i.e. after the previous matching position).

vii. If found at a position P', then match next letters of the target document and increase L' respectively until a mismatch or end of any of the two documents occurs.

viii. If current length L' of common pattern is greater than previous length L, then update the value of length L with L' & position P with P'.

ix. Repeat the step (vi) to (viii) until end of key document (or target document) occurs.

x. Now first longest sequence of length L (i.e. a prefix of target document) has been found in the key document from position P to P+L-1.

xi. Note down the first ordered pair as (P, L).

xii. Now repeat entire pattern matching process for remaining document i.e. starting from the very next position in the target document up to which the common sequence has been found.

xiii. Create the ordered pairs until entire target document is finished, (i.e. End of target document occurs).

xiv. If a strange element comes in the target document and key document fails to find a common sequence of that element then consider the position P in the key document as -1 (or any negative number for that matter.) It indicates that no match found in the key document at any real position. Put the element itself (or certain transformed value for security) as second parameter of the ordered pair, because it has no reference in the key document. Length of the common sequence is no longer needed in such cases as it is always 1.

C. Algorithm

```
procedure compress_and_cipher()
  initialize i=base=offset=length=begin=0
  target[]; array of target document elements
  key[]; array of key document elements
  while target[base] do
    while target[base+i] AND key[offset+i] do
      while target[base+i]==key[offset+i] do
        i++
      end while
      if i>length do
        begin=offset
        length=i
      end if
      offset++
      i=0
    end while
    if length==0 do
      gen_op(-1, target[base])
      base++
    else
      gen_op(begin, length)
      base+=length
    end while
    length=offset=0
  end while
end procedure
procedure gen_op(P, L)
  store op{(P, L)}
  return
end procedure
```

D. Illustration

Key: *"a quick brown fox jumps over the lazy dog."*

Target: *"oh! the fox is so quick."*

Start with the first character of the target *'o'*, it matches with 11th character of the key. But second character of the target *'h'* doesn't match with the 12th character of the key. Till now; position P=11 & length L=1. Again search for the 1st letter *'o'* of the target in the remaining key. Next match occurs at 16th position but again it fails to match next character. Further matches occur at 25th & 40th positions but both fail to match the next character. Hence the first ordered pair is *{(11,1)}*.

Remaining target stats with *'h'* its 1st match occurs at 31st position in the key but again next character mismatches. No other *'h'* exists in the remaining key. Again the length of the longest common sequence is 1. It generates ordered pair *(31,1)*. Hence the ordered pairs are *{(11,1) (31,1)}*.

Next character of the target is *'!'*, which is a strange element for the key i.e. no match exists in the entire key. So the position is taken as -1 and instead of length character *'!'* itself is taken as second parameter of the ordered pair. Hence the ordered pairs are *{(11,1) (31,1) (-1,!)}*.

Next character of the target is a *'white space'* which matches at 2nd, 8th, 14th, 18th, 24th, 29th, 33rd & 38th positions of the key. All except 29th fail to match the next character. But it matches from 29th to 33rd in the key which produces L=5. Hence ordered pairs are *{(11,1) (31,1) (-1,!) (29,5)}*. This process keeps on until the target lasts.

Ordered Pairs: *{(11,1) (31,1) (-1,!) (29,5) (15,4) (5,1) (23,2) (23,1) (11,1) (2,6) (42,1)}*

IV. DECOMPRESSION & DECIPHERING

Decompression & deciphering is an easier and simpler process than compression & ciphering and has no time complexity (as no comparison is needed).

A. Objective

To regenerate the target document using key document and set of ordered pairs.

B. Method

i. Take the first ordered pair and read its first parameter (i.e. Position), find out the position P in the key document.

ii. Read the second parameter L of the ordered pair (i.e. Length).

iii. Copy the substring of key document from position P to P+L-1 into a new blank document.

iv. Take the next ordered pair and according to the parameters, find out the sub-sequence and append it into the freshly taken document.

v. If first parameter of the ordered pair is -1, then simply append the second parameter of the ordered pair into the fresh document.

vi. Repeat the process until all the ordered pairs last.

vii. Resulting fresh document is decompressed & deciphered form of the actual target document.

C. Algorithm

```
procedure decompress_and_decipher()
  set op; set of ordered pairs
  initialize temp=NULL
  initialize new_doc=NULL
  for all op do
    if op->position is -1 do
      temp=op->length
    else
      temp=sub_str(op->position, op->length)
    end if
    new_doc+=temp
  end for
end procedure
procedure sub_str(P, L)
  temp=NULL
  for i=P to P+L-1 do
    temp+=key[i]
  end for
  return temp
end procedure
```

D. Illustration

Key: *"a quick brown fox jumps over the lazy dog."*

Ordered Pairs: *{(13,1) (11,1) (-1,,) (8,2) (4,1) (30,1) (29,5) (39,3) (2,1) (5,1) (23,2) (23,1) (11,1) (33,5) (42,1)}*

Start with the first ordered pair *(13,1)*; it implies that the target begins with a subsequence of length L=1, and position 13 in the key. Hence the subsequence is *"n"*.

Next ordered pair is *(11,1)*; it implies that the next subsequence is again of length L=1 and position 11 in the key. Hence the subsequence is *"o"*. Concatenate it with the previous subsequence. Updated target prefix becomes *"no"*.

Next ordered pair is *(-1,,)*; it's position parameter is -1, which indicates that there is some strange element which has no match in the key. So the subsequence generated by this ordered pair is the second parameter itself i.e. *","*. Hence the updated prefix of the target is *"no,"*.

Similarly next ordered pair *(8,2)* generates a subsequence of length L=2 and start position P=8 as *" b"*. Hence the updated target prefix is now *"no, b"*.

This process keeps on until the last ordered pair of the set. And finally generates the whole target.

Target: *"no, but the dog is so lazy."*

V. DISCUSSIONS

1) Larger the key document, better the degree of compression, but more the time complexity.

2) For improved security this method can be applied in *Cascade* mode (i.e. the key document can further be ciphered with reference to any other key document).

3) Larger the common patterns & fewer the overlapping sequences, lesser the chances of deciphering by guessing.

4) Patterns are sequential in the target document but are independent in the key document (i.e. these may be adjacent, apart or over lapping).

5) To use a fine document as key document is a good choice for document transmission if it is available at both the ends. But if key document also needs to be transmitted then a virtually fine document is preferred.

6) To enhance security; if multiple identical common sequences are found in the key document then instead of taking first one, any one of them can be chosen randomly, whenever such pattern comes in the target document.

7) An ideal document of a particular document type must be unique. Otherwise one ideal document will fail to generate single ordered pair for entire other ideal document.

8) An ideal document has undefined start point as well as end point.

9) It may take extremely large time to find out a single pattern for the whole target document if ideal document is taken as a key document.

10) An ideal document can compress any document of its category to the best possible degree of compression (i.e. in only a single ordered pair).

11) If the target & key documents are identical, then the set of ordered pairs will be a single set. The first parameter of the ordered pair will be 0 & second parameter will be the length of the document.

12) In the traditional Steganography techniques actual data are injected into the carrier file, which causes the distortion in the carrier file and also limits the carrying capacity of the carrier file. While in this technique of hiding data within the data actual document is only being referred by the carrier (or key) document, which causes no distortion and leads to unlimited carrying capacity of the carrier document.

VI. CONCLUSION

Document Compression and Ciphering Using Pattern Matching Technique can be used for secure data transmission over the internet. Proper key documents can provide extreme compression which saves lots of bandwidth and results in faster data transmission. Heavy web portals can use *Virtually Fine* documents of strongly associated pages to deliver maximum information by transmitting minimum bytes of data. It is yet another Steganography Technique.

REFERENCES

[1] James C. Judge, "Steganography: Past, Present, Future," UCRL-ID-151879 December, 2001. https://e-reports-ext.llnl.gov/pdf/245799.pdf

[2] Boyer R.S., Moore J.S., "A fast string searching algorithm," Communications of the ACM, Vol. 20, No. 10, pp. 762-772, 1977.

[3] Knuth D.E., Morris J.H., Pratt V.R., "Fast pattern matching in strings," SIAM Journal on Computing, Vol. 6, No. 2, pp. 323-350, 1977.

[4] P.D. Michailidis & K.G. Margaritis, "On-line String Matching Algorithms: Survey and Experimental Results," IJCM 2001, VOL 76; PART 4, pages 411-434