This project aims to hide a secret Persian text in a Persian text as carrier by using Unicode.

The project is written with C# 2008 in two classes. One for steganography and another is for steganalysis.

There are 3 functions in steganography class:

- ProduceBinCode: Gets the secret message in text as input and returns its binary equivalent.
- SetKey: Returns the key based on the length of binary equivalent string.
- SetStego: Gets the binary equivalent string and the original text and returns the stego text.

➢ ***How the functions are programmed:***

ProduceBinCode

The secret message includes of Persian letters and space character. There are 32 letters in Persian script. The letter “ا” comes in two forms “ا” and “آ”. Including space character and two forms of “ا”, we have 34 characters to code. With considering 6 binary digits per character, we can code 2^6=64 characters. In future we are able to add more characters in secret message.

These characters are sorted in the script array. This array starts with “آ” following by “ا” and then rest of characters comes by their alphabetic order. The last character in this array is space. The binary equivalent for each character is its order number.

script = {'آ','ا','ب','پ','ت','ث','ج','چ','ح','خ','د','ذ', 'ر', 'ز', 'ژ', 'س', 'ش', 'ص', 'ض', 'ط', 'ظ', 'ع', 'غ', 'ف', 'ق', 'ک', 'گ', 'ل', 'م', 'ن', 'و', 'ه', 'ی',' '};

❖ The binary conversion algorithm works in the following steps:
    1. The function reads the characters from secret message one by one and finds the character order number in the script array.
    2. The order number converts to a 6 digits binary number.
    3. The function works until the secret message end.

As an example, the binary equivalent for “در” word is “001010001100”. “001010” is equivalent for “د” and “001100” is equivalent for “ر”. The order number for “د” and “ر” is 10 and 12 in the script array respectively.

SetStego: (Main steganography function)

We consider 4 arrays with names: “zz”, “zo”, “oz” and “oo”. “z” represents zero and “o” represents one. In each turn of running the function, two binary strings are being hided in one letter. Based on the observation from all binary equivalents which has gained based on ProduceBinCode function, the number of “00” is 35, “01” is 33, “10” is 18 and “11” is 16, then the number of “00” is the maximum.

Based on some evaluations on Persian texts, the characters with more frequency have been chosen to hide “00”. The rest of arrays are also sorted based on character frequency. These arrays are shown below. The numbers in the arrays are Unicode codepoints for each character.

```
zz = { 1570, 1575, 1583, 1584, 1585, 1586, 1688, 1608, 32, 8204 };
zo = { 1705, 1711, 1604, 1605, 1606, 1607, 1740, 1587,1600 };
oz = { 1662, 1578, 1579, 1580, 1670, 1593, 1594, 1601,1602 };
oo = { 1576, 1588, 1581, 1582, 1589, 1590, 1591, 1592,8205 };
```

To have more throughput for data conceal, in addition to alphabetic letters, we use kashida and zwj and zwnj characters too. The "zz" array consists of isolated letters (the ones don't connect to their previous or next character).

From Wikipedia, The zero-width non-joiner (ZWNJ) is a non-printing character used in the computerization of writing systems that make use of ligatures. When placed between two characters that would otherwise be connected into a ligature, a ZWNJ causes them to be printed in their final and initial forms, respectively. This is also an effect of a space character, but a ZWNJ is used when it is desirable to keep the words closer together or to connect a word with its morpheme. The zero-width joiner (ZWJ) is a non-printing character used in the computerized typesetting of some complex scripts such as the Arabic script or any Indic script. When placed between two characters that would otherwise not be connected, a ZWJ causes them to be printed in their connected forms.

We can place "ZWJ" character in every of 3 arrays which are for connected letters, then we place it in "oo" group and place "ZWNJ" character in the "zz" array which is for non-connected letters. The observation on kashida character with hexadecimal code "0640" shows if this character comes after marker characters (we explain it later) makes no change in the text appearance. Therefore, we place this character in "zo" array.

The data hides in each group letter has been shown in the table. For example when we have "00" to hide, it will be hided in the letters of group "zz".

Setting the markes

In order to detect carrier letters (the letters which has secret data), a marker is needed. Transparency is one of the main items in steganography and stego text should be look the same as original text, therefore we benefit from Unicode properties to mark the carrier letters. As we explained about "zwj" and "zwnj" characters earlier, we use them as markers. The marking process works as below:

In every turn of running function two binary digits from secret message are selected and the function brows the carrier text character by character. If we have two "00" to hide, a letter from zz array should be found in the carrier text for hiding data. Since the letters in this array don't connect to their next letter, no condition has been set in this situation. To mark the letters in this array, two "zwnj" characters with hexadecimal "200c" code will be inserted after the carrier letter.

If the carrier letter is in the other arrays, since these letters connect to their next letter, we need to check the next character. If the next character is an alphabetic letter, there is a connection between these two letters, and two "zwj" characters with "200D" hexadecimal code will be inserted. Since this character is for connecting letters and the carrier letter is connected to its next letter, inserting these two characters does not change the look of original text. If the next character is a nonalphabetic character, we insert two “zwnj” characters. This change doesn’t have impact on original text as well. This method increases the throughput. We can hide data in any situation.

There is an exception in this case, if there is kashida character with “0640” codepoint after the carrier letter, we skip this letter. The observation shows if “zwj” or “zwnj” characters insert between a letter and kashida, this change the original text noticeably.

Since there might be other “zwj” and “zwnj” characters in the text and get mistaken by our markers, we use two of these characters as marker. Setting two characters makes the method more accurate.

In summary, the steganography method works in the following order:

1. The secret message converts to a binary string.
2. The function reads two characters of binary string in each time, and finds the associated array for these two binary letters. The carrier text will be browsed to find a letter in the associated array and this letter will be the carrier letter.
   - ❖ If the binary string is in the “zz” array, without considering any conditions, two zwnj characters will be inserted after the carrier letter.
   - ❖ If the binary string is in the other arrays and the next character is not kashida, the next character will be checked:
     - If the character is an alphabetic one, insert two “zwj” after the carrier letter.
     - Else insert two “zwnj” characters.

This function continues running until the end of secret binary string or original text.

Since in every turn of running steganography function, 2 binary code are being hided. Binarymessage length/2 will return as key.

Steganalysis function

This function works as below:

- The function gets the key and stego text as inputs.
- A counter has been set to 0
- The function brows the stego text to find two “zwnj” or two “zwj”.
- When these markers have been founded, the letter before the markers is the carrier letter.
- The function extracts the letter and compares it with the arrays. When finding the associated array, two binary digits of secret message will be extracted.
- Counter= counter+1
- These steps repeat to counter=key

Since there are 6 binary digits for each letter, after extracting the secret message in binary, characters are separated 6 by 6 and the decimal equivalent for this binary text will be acquired and the alphabetic character will be extracted based on script array.

Evaluation

This method worked on some office word files files with doc and docx formats. In these experiments, the main and secret message length were variable. The result has been shown in below table.

Original Text length(Characters num)	**Secret Message length(Characters num)**	**Throughput(secret len/original text len)*100**
5712	336	5.88
5242	318	6.06
511	28	5.47
784	32	4.08
3821	214	5.60
1070	59	5.51
1509	79	5.23
769	35	4.55
1646	71	4.31
2136	126	5.89
Average Throughput		5.25

Positive Points

- Because of using Unicode, this method is persistence against changing type, size and font
- This method is persistence against line distance change, increase or decrease indentions in paragraphs.
- Persistence against changing the text alignment from left to right or inverse.
- Adding a word inside text doesn’t destroy the secret message
- Inserting other items such as pictures, charts etc. doesn’t harm the stego text.

This method can met transparency because of using Unicode and it has acceptable throughput and persistent against changes.