Bifid Cipher

Polybius Square Cipher

Keane J. Moraes

1. History

The Bifid cipher was invented by amateur cryptographer Felix Delastelle. It was first presented in *Revue du Génie civil* in 1895. He was best known for his Bifid, Trifid and Four-Square cipher along with variant on the Playfair cipher.

2. Modern Usage

The Bifid cipher has never entered use in any military or government organization and is implemented only among hobbyists.

3. Encryption and Decryption Algorithms

Encryption using the Bifid cipher involves use of a standard Polybius square with transposition^{I} followed by fractionation^{II} to achieve diffusion^{III}.[1] A mixed alphabet Polybius square may be used with slight modifications to the code. One may also use a key to generate a mixed alphabet Polybius Square. This is done by our `generateCustomKey()` method from the Polybius class.

3.1. Encryption

Our encryption will take place in 3 steps. We will define an 'intermediate String' to store out Polybius Sqaure coordinates. We will call this String our 'coordinate transcribe'. For simplicity, we will be operating only with uppercase letters but one can tweak the formula and add if statements to account for lower case letters too.

STEP 1: Convert the plaintext into Polybius Square coordinates
In order to convert our plain text to coordinates we will use the same technique as we did in the decryption algorithms of Polybius Square. However, we cannot store the both the row and column coordinates in one String as we have to concatenate the two later to achieve fractionation. Let us try to work through an example to illustrate this.

Suppose we want to encrypt the word CRYPTOGRAPHY using the standard Polybius square. Refer to the Encryption section of Polybius Square cipher to see how this is done. The diagram shows how we split the coordinate transcribes of the characters.

	C	R	Y	P	T	O	G	R	A	P	H	Y
r	1	4	5	3	4	3	2	4	1	3	2	5
c	3	2	4	5	4	4	2	2	1	5	3	4

[1] From the Wikipedia page on the Bifid Cipher

Let E_t be the cipher text.
P_t be the plain text having length n.
R_t be the row coordinates.
C_t be the column coordinates.
$R_t = R_1R_2R_3 \dots R_n$
$C_t = C_1C_2C_3 \dots C_n$
The i^{th} character of the row and coordinate transcribe is -
Let K be the index of the character in the custom key.[2]

$$R_i = \left(K \div 5 + 1\right)$$

$$C_i = \left(K \mod 5 + 1\right)$$

(Additionally if that plaintext character is 'J', it is indexed as 'I').

STEP 2: Concatenate the row and column coordinates into one
The Row coordinates are first followed by Column coordinates.

STEP 3: Convert the new coordinates into letter using Polybius Square
After having concatenated both the coordinates into one String, we implement the Polybius Square encryption algorithms for consecutive letters.

14	53	43	24	13	25	32	45	44	22	15	34
D	X	S	I	C	K	M	U	T	G	E	O

Definitions

I - Transposition : A transposition of the plain text is set of instructions work on the position of the plain text rather than the text itself. Encryption is done by operating on the positions of the characters in the plain text. For eg. Rail fence cipher and Route cipher are common transposition ciphers.

II - Fractionation : is a technique where a plaintext is split up so that it is represented by two or more symbols.[a]. In the Bifid cipher, the fractionation is achieved when we place the plaintext letters on a grid and replace it with the corresponding row and column coordinates.

III - Diffusion - is a security term coined by Claude Shannon in his report "*A Mathematical Theory of Cryptography*". Diffusion means that if we change even a single bit of the plain text then half the bits of the cipher text should change and vice versa.[b]

[a] Crypto Corner : Fractionating Ciphers
[b] Confusion and Diffusion

3.1.1. Code Implementation

The code implementation of the encryption algorithm also takes place in the 3 steps listed above.

STEP 1:

```
47    String result = "", rowNumbers = "", columnNumbers = "";
48    plainText = (plainText+" ").toUpperCase();
```

[2] Note that indexing starts with 0

Here the variables `rowNumbers` and `columnNumbers` store the row and column coordinates of the letters of the plaintext respectively.

```
50    // S1 : ENCODING LETTERS INTO POLYBIUS SQUARE COORDINATES
51    for (int i = 0; i < plainText.length(); i++) {
52      char character = plainText.charAt(i);
53      if (character < 65 || character > 90)
54        continue;
55      if (character == 'J')
56        character = 'I';
57      int letterNumber = key.indexOf(character);
58      rowNumbers += (letterNumber / 5 + 1) + "";
59      columnNumbers += (letterNumber % 5 + 1) + "";
60    } // for loop - i
```

In the `for`-loop, for each character we check whether it is a valid uppercase character. If it isn't then it is just ignored. If the character is 'J' then it is reassigned to 'I'. This is a standard Polybius square cipher encryption technique. The only difference is that the code does not append the row and the column numbers together, rather it stores them in different strings.

STEP 2:

```
66      // S2 : CONCATENATING THE ROW AND COLUMN COORDINATES
67      rowNumbers += columnNumbers;
```

Just as stated in STEP 2 of the encryption algorithm, we concatenate the row and the column coordinates. The value of the concatenation is stored in `rowNumbers`.

STEP 3:
Now we must use the Polybius decryption algorithm to encrypt the contents of `rowNumbers`. The code from line 69 is the same as the code implementation of the Polybius Decryption algorithm. The result of this is then returned.

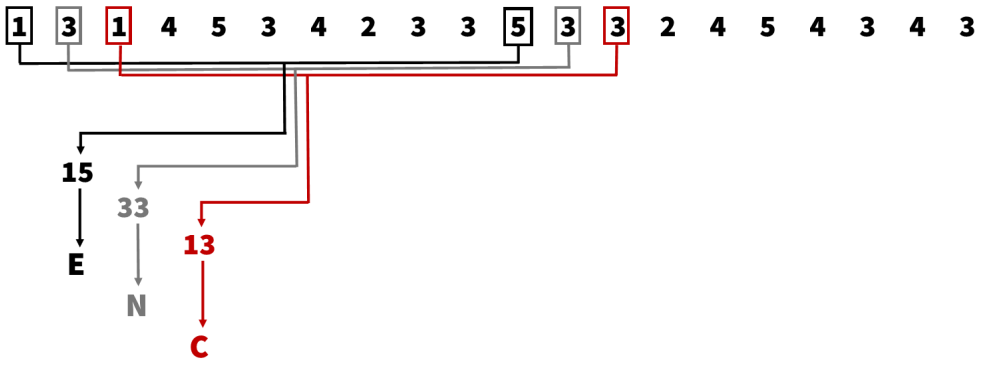
3.2. Decryption

Decryption, just like encryption, is a multi step process. It will take place in 2 steps. Once again we define an intermediate String called the 'coordinate transcribe'.

STEP 1: Convert the encrypted text into Polybius Square coordinates
Using the Polybius Square encryption algorithms, we convert the cipher text into our coordinate the row and column coordinate transcribes.
Let C_t be the coordinate transcribe.
$C_t = C_1C_2C_3C_4 \ldots C_n$ where n is the length of the coordinate transcribe.

C	D	X	R	N	X	M	U	S	S
13	14	53	42	33	53	32	45	43	43

STEP 2: Isolate the i^{th} and the $(mid + i)^{th}$ numbers to convert back into letters
We define a variable called mid which stores half our coordinate transcribes length. Now we pair up (concatenate) each j^{th} character from the start of the string and j^{th} character from the midpoint of the string to be our coordinates for our conversion back into letters. The below diagram depicts this in action.



We will proceed with a standard Polybius decrypt after we have gotten out the coordinates.
Let P_t be the plain text.

$K = S_j S_{mid+j}$ be the integer coordinates that we will use to convert back into letters (note that $S_j S_{mid+j}$ means concatenation not multiplication). The i^{th} character of the plain text is -

Let K be the index of the encrypted text character in the custom key.[3]

$$P_t \text{ is the } \left[(S_j - 1) \times 5 + (C_{mid+j} - 1)\right]\text{-th letter of the key}$$

(This formula assumes the indexing starts with 0).

3.2.1. Code Implementation

Following the same structure as the code implementation for encryption, this is structured into 2 steps

STEP 1:

```
// S1 : CONVERSION OF ENCRYPTED TEXT INTO COORDINATES
String result = "", intermediateLetterNumbers = "";
intermediateLetterNumbers = Polybius.encrypt(encryptedText, key);
key = generateCustomKey(key, "ABCDEFGHIKLMNOPQRSTUVWXYZ");
```

Here we use the Polybius Encryption process to convert the letters of the encrypted text into Polybius Square coordinates. This is identical to the code implementation of the Polybius encryption process. The `intermediateLetterNumbers` stores the Polybius Square coordinates of the encrypted text. The unique key is extracted from the seed.

STEP 2:

```
int midPoint = intermediateLetterNumbers.length()/2;
```

This is the midpoint variable that stores the *mid* as described in the decryption algorithm process. You can verify that the length of `intermediateLetterNumbers` is always even.

```
for (int i = 0; i < midPoint; i++) {
  int letterNumber = Integer.parseInt(intermediateLetterNumbers.charAt(i) + "" +
  intermediateLetterNumbers.charAt(midPoint + i));
```

This step is the crucial step in making the decryption work. Here we extract the i^{th} character and then the $(\text{mid} + i)^{th}$ character and form a Polybius Square coordinate.

```
  int rowNumber = letterNumber / 10, columnNumber = letterNumber % 10;
  result += key.charAt(--rowNumber * 5 + --columnNumber);
```

Notice that the code above is identical to that in Polybius decryption where we use the coordinate to find the letter in the `key`.

[3] Note that indexing starts with 0

4. Further Reading

Cryptanalysis

Refer to the Practical Cryptography webpage for additional information into the cryptanalysis of the Bifid.

Fractionating Ciphers

Crypto Corner's article on Fractionating Ciphers is a useful resource.

Refer to this Crypto SE post for more information - In cryptography, what is "fractionation"?