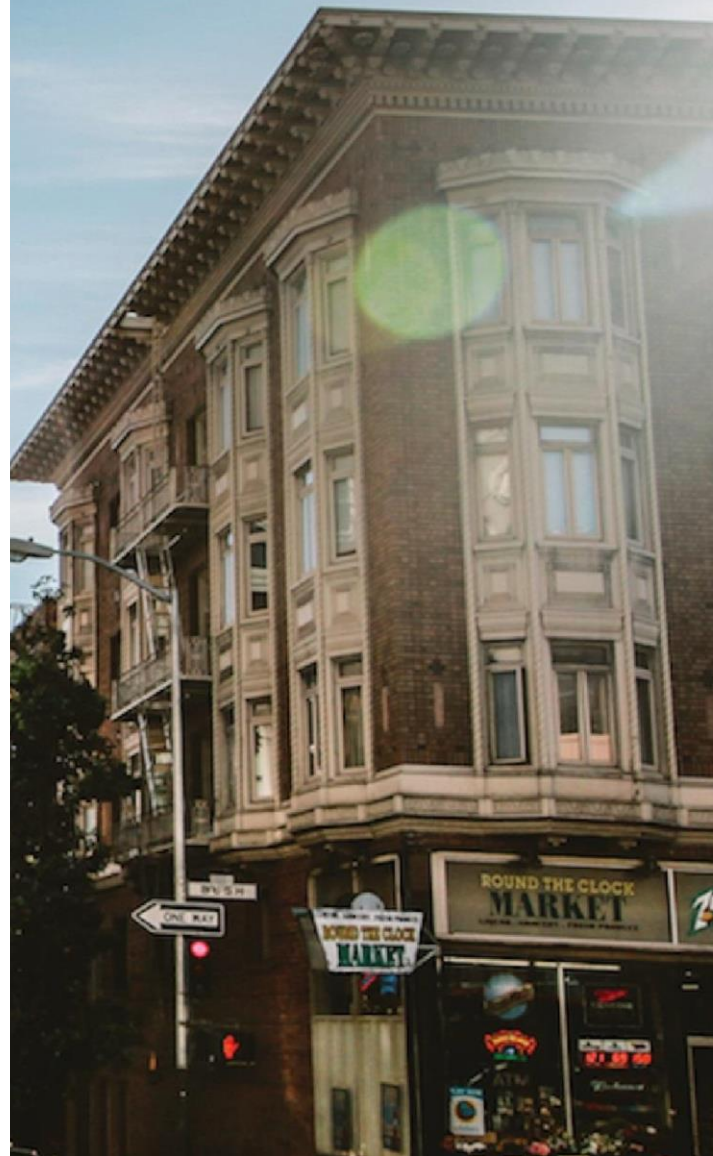


# ENHANCEMENT OF SYMMETRIC KEY CRYPTOGRAPHY USING RUBIK'S CUBE

---

20BCN7028 – D. Venkata Karthik  
20BCE7363 – KVS Sumanth Guptha  
20BCE7404 - K Praneeth

---



# Abstract

The current scenario is such that the assurance of security in large open networks has become the need of the hour. With increase in the rate of crimes, one needs to take precautions to protect the data in an efficient manner from all possible attacks. Basically for this need we have undertaken the task of providing such a secured package, which also provides secured data transmission environment to the user.

This all is possible using cryptography. Explaining each aspect in detail as follows, beginning with Cryptography and Hill cipher using Rubik's Cube. Hill cipher is a polygraphy substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. Rubik's cube is a puzzle in the form of a plastic cube covered with multi-coloured squares, which the player attempts to twist and turn so that all the squares on each face are of the same colour.

The main objective of this project is to enhance the working of hill cipher using a simple puzzle. To protect the key of hill cipher from various cryptanalyst attacks, we are going to use the number of possible outcomes on a Rubik's cube as a key to Hill cipher to encrypt the plain text. This results in widen the range of keys in hill cipher and also lessen the possibilities to decrypt the plaintext.

# Software requirements

Java technology is both a programming language and a platform. The Java programming language is a high-level language that can be characterized



By all of the following buzzwords:

- Simple
- Architecture neutral • Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The IDE used in this project is Eclipse IDE.

# Hill cipher

- We define the inverse  $M^{-1}$  of a square matrix  $M$  by the equation  $M(M^{-1}) = (M^{-1})M = I$ ,
- Where  $I$  is the identity matrix.  $I$  is a square matrix that is all zeros except for ones along the main diagonal from upper left to lower right. The inverse of a matrix does not always exist, but when it does, it satisfies the preceding equation.
- For example:

$$A = \begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix} \quad A^{-1} \bmod 26 = \begin{pmatrix} 9 & 2 \\ 1 & 15 \end{pmatrix}$$

$$\begin{aligned} AA^{-1} &= \begin{pmatrix} (5 \times 9) + (8 \times 1) & (5 \times 2) + (8 \times 15) \\ (17 \times 9) + (3 \times 1) & (17 \times 2) + (3 \times 15) \end{pmatrix} \\ &= \begin{pmatrix} 53 & 130 \\ 156 & 79 \end{pmatrix} \bmod 26 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned}$$

- For encryption of hill cipher we need to multiply the plaintext with the key to get the resultant ciphertext. That is,
- Input: Plaintext that has to be converted into ciphertext.
- A KEY to encrypt the plain text
- Output: Ciphertext
- We have a simple formula for encryption  
 $C = K \cdot P \bmod 26$
- Where  $C$  is ciphertext,  $K$  is the key,  $P$  is the plain text vector.

- The KEY is generally in format of  $n \times n$  matrix and the plain text is divided into  $n \times 1$  vectors to hide the information of plaintext and transform into the text into ciphertext.
- For demonstration, let's take an example:

Plain text: "exam"

Key:

$$\begin{bmatrix} 7 & 8 \\ 11 & 11 \end{bmatrix}$$

$$C = P * K \bmod 26 =$$

$$C = \begin{bmatrix} 7 & 8 \\ 11 & 11 \end{bmatrix} \begin{bmatrix} 4 \\ 23 \end{bmatrix} \bmod 26$$

$$C = \begin{bmatrix} 4 \\ 11 \end{bmatrix} = \begin{bmatrix} e \\ l \end{bmatrix}$$

For, first column

$$C = \begin{bmatrix} 18 \\ 2 \end{bmatrix} = \begin{bmatrix} s \\ c \end{bmatrix}$$

For, Second column

Hence, the ciphertext is "elsc".

To deal with decryption, first we need to find the inverse of a key matrix such that,

- Find the adjacent matrix of the given key matrix  $K_{adj}$  = Adjacent matrix key cipher text
- Find the determinant of the key matrix  
 $77 - 88 = -11$
- 3) Find the modulo of the determinant with 26  
 $-11 \bmod 26 = 15 = d$

- Find the inverse number of the above result:

$$d \times d' = 1 \pmod{26}$$

$$x \times d' = 1 \pmod{26}$$

$$d' = 7$$

- Any negative numbers in  $K_{adj}$  should be added by 26 and then the whole matrix is multiplied by  $d'$ .

$$\begin{bmatrix} 11 & -8 \\ -11 & 7 \end{bmatrix} \longrightarrow \begin{bmatrix} 11 & 18 \\ 15 & 7 \end{bmatrix} \times 7$$

$K' =$  negative numbers in  $K_{adj}$

- Now, this is our new key matrix. Substituting all the values in the decryption formula, we get the required plain text.

$$C = \begin{bmatrix} 77 & 126 \\ 105 & 49 \end{bmatrix} \begin{bmatrix} 4 \\ 11 \end{bmatrix} \pmod{26}$$

- Formula for decryption of hill cipher is:

$$P = K^{-1}C \pmod{26} = KK^{-1}P = P$$

Substituting all the values in the decryption formula

$$P = \begin{bmatrix} 4 \\ 23 \end{bmatrix} = \begin{bmatrix} e \\ x \end{bmatrix}$$

For column 1,

$$P = \begin{bmatrix} 0 \\ 12 \end{bmatrix} = \begin{bmatrix} a \\ m \end{bmatrix}$$

For column 2.

- Hence, the decryption is successful and the original message has been retrieved.

# PlayFair cipher:

- In this algorithm, an alphabets table of 5×5 grid is created as a key for encrypting the plaintext.
- Each of the 25 alphabets must be unique, since we have 26 letters in English alphabet, one letter (usually J) is omitted from the table.
- If the plaintext contains J, then it is replaced by I. The sender and the receiver decide on a particular key.
- In a key table, the first characters (going left to right) in the table is the phrase, excluding the duplicate letters. The rest of the table will be filled with the remaining letters of the alphabet, in natural order.
- To encrypt a message using Playfair Algorithm, first, a plaintext is split into pairs of two letters. If the message contains an odd number of letters, then a letter X is added to the last letter, for example, the message “odd” will be written as      -od dx.
- Depending on the location of each two letters in the matrix key, encryption will do according to the rules below:
  1. If both the letters are in the same column, take the letter below each one (going back to the top if at the bottom)
  2. If both letters are in the same row, take the letter to the right of each one (going back to the left if at the farthest right)
  3. If neither of the preceding two rules are true, form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

# Methodology of Playfair:

**Encryption phase:** We must now split the plaintext up into digraphs (that is pairs of letters). On each digraph we perform the following encryption steps:

- 1- If the digraph consists of the same letter twice (or there is only one letter left by itself at the end of the plaintext) then insert the letter "X" between the same letters (or at the end), and then continue with the rest of the steps.
- 2- If the two letters appear on the same row in the square, then replace each letter by the letter immediately to the right of it in the square (cycling round to the left hand side if necessary).
- 3- If the two letters appear in the same column in the square, then replace each letter by the letter immediately below it in the square (cycling round to the top of the square if necessary).
- 4- Otherwise, form the rectangle for which the two plaintext letters are two opposite corners. Then replace each plaintext letter with the letter that forms the other corner of the rectangle that lies on the same *row* as that plaintext letter (being careful to maintain the order).



### Example:

```
Plaintext:
MAGICMIND
Key:
COMPUTER

-----Key Matrix-----
      C      O      M      P      U
      T      E      R      A      B
      D      F      G      H      I
      K      L      N      Q      S
      V      W      X      Y      Z
-----
Encrypted text:
-----
PR HD OP GS GV
```

Where,

- MAGICMIND is an odd numbered word, so last word is quoted as “X” to make it even numbered.
- Using Key matrix, Now we encrypt our plaintext: MAGICMIND as:  
MA - PR GI - HD CM-OP IN-GS DX – GV (Applied the rules of encryption)

**Decryption phase:** Decryption is nearly identical to the encryption process, except for rules 2 and 3 we must take the letters to the left and above respectively. Also, we remove any extra "X" in the decrypted text to reveal the final plaintext.

#### Rules of Decryption:

- 1- Two ciphertext letters in the same row of the matrix are each replaced by the letter to the left, with the last element of the row circularly following the first.
- 2- Two ciphertext letters that fall in the same column of the matrix are replaced by the letters above, with the bottom element of the column circularly following the top.
- 3- Otherwise, each ciphertext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other ciphertext letter.
- 4- Following these rules, the plaintext becomes 'MAGICMIND'. {x is neglected or removed at the end since it's a filler letter}.

#### Example Continuation:

- We shall decipher the ciphertext "PRHDOPGSGV " which has been encrypted using the keyword example. Firstly we must generate the Polybius Square which we are using, as shown in below:

C	O	M	P	U
T	E	R	A	B
D	F	G	H	I
K	L	N	Q	S
V	W	X	Y	Z

- Next step is splitting the encrypted text into:

PR	HD	OP	GS	GV
----	----	----	----	----

- Applying the rules of decryption for the well divided encrypted text using Key Matrix. Our Plaintext is retrieved.

```
Plaintext:
MAGICMIND
Key:
COMPUTER

-----Key Matrix-----
      C      O      M      P      U
      T      E      R      A      B
      D      F      G      H      I
      K      L      N      Q      S
      V      W      X      Y      Z

-----
Encrypted text:
-----
PR HD OP GS GV
-----
Decrypted text:
-----
MA GI CM IN DX
-----
```

# Rubik's Cube

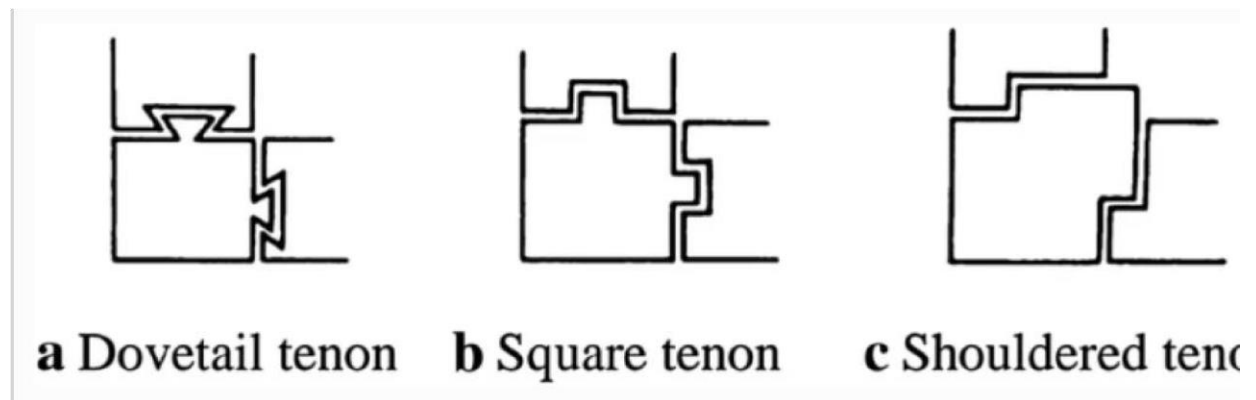
Rubik's Cube is a widely popular mechanical puzzle that has attracted attention around the world because of its unique characteristics. As a classic brain-training toy well known to the public, Rubik's Cube was used for scientific research and technology development by many scholars.

This paper provides a basic understanding of the Rubik's Cube and shows its mechanical art from the aspects of origin and development, characteristics, research status and especially its mechanical engineering design, as well as making a vision for the application in mechanism.

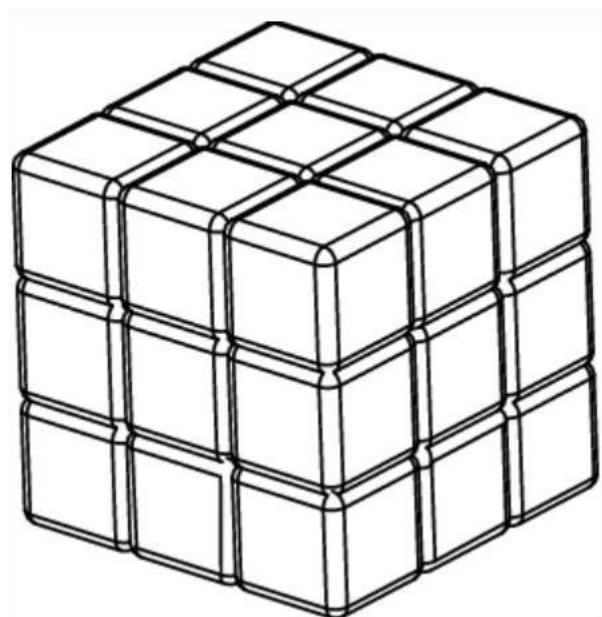
First, the invention and origin of Rubik's Cube are presented, and then the special characteristics of the cube itself are analyzed. After that, the present researches of Rubik's Cube are reviewed in various disciplines at home and abroad, including the researches of Rubik's Cube scientific metaphors, reduction algorithms, characteristic applications, and mechanism issues.

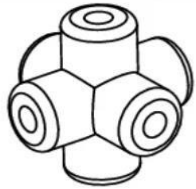
### Structure of Rubik's cube:

- Rubik's Cube were based on mortise and tenon connections between pieces. The mortise and tenon methods can be used so that the 26 exterior pieces of the Cube are held together with no central piece at all. This connection is difficult to make with sufficient accuracy so that the faces will turn easily [18]. The mortise and tenon movements are shown below:

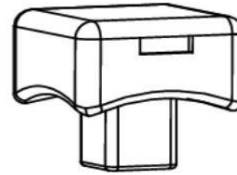


- The standard  $3 \times 3 \times 3$  cube consists of 26 unique miniature cubes, also called "pieces" or "cube lets" including a center shaft frame, 6 center pieces, 8 corner pieces, and 12 edge pieces. The various types of pieces of Rubik's Cube are shown below:

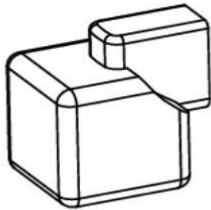




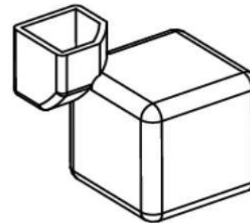
**a** Center shaft frame



**b** Center piece

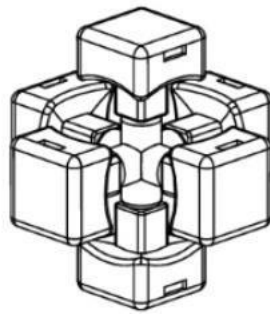


**c** Edge piece

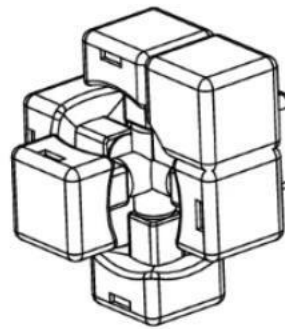


**d** Corner piece

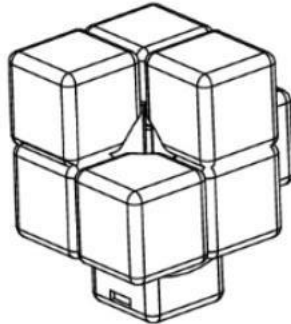
- The constraints on the edges are formed by structural restrictions and the force locking of two adjacent centers. The constraints on the corner piece are formed by structural restrictions and the force locking of three adjacent edges.



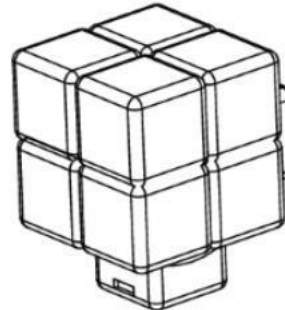
**a** Connection between six centers and center shaft frame



**b** Connection between two centers and one edge

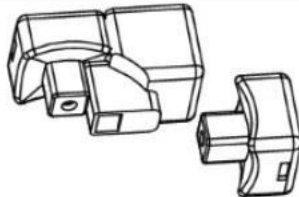


**c** Connection between three centers and three edges

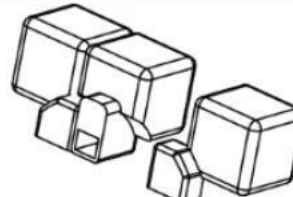


**d** Connection between three edges and one corner

- The contact modes between the centers and edges, and the edges and corners, are shown:



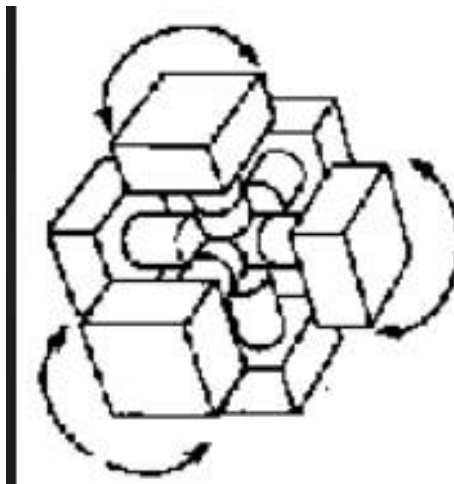
**a** Contact mode between centers and edges



**b** Contact mode between edges and corners

## Mathematics of Rubik's cube:

- The original  $(3 \times 3 \times 3)$  Rubik's Cube has six faces. Initially, each face has the same color, and each face has nine small outer surfaces. They are 54 outer surfaces in total. Every side of Rubik's Cube is composed of diversiform color pieces after the random rotation of different sides several times.
- After disassembling the cube, the structure is as shown:



- There are 12 edge piece positions. Each time when we insert an edge piece in its position, the number of possibilities after insertion at an edge piece gets reduced by 1.

Therefore,

The number of possibilities on an edge positions =

$$12 \cdot 11 \cdot 10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 12!$$



- Similarly, there are 8 corner piece positions. Each time when we insert in an corner piece position. The number of possibilities after insertion at an edge piece gets reduced by 1.

Therefore,

The number of possibilities on an corner positions =  $8*7*6*5*4*3*2*1$   
 $= 8!$

- Total number of possibilities that a cube can be afforded after insertion is:

$$12! * 8! = 19,313,344,512,000$$

- Considering the flips of edge pieces =  $2^{12}$
- Considering the twists of corner pieces =  $3^8$
- Total number of possibilities after considering flip positions is:

$$12! * 8! * (2^{12}) * (3^8)$$

- However,

1. No single edge flip =  $2^{12}/2 = 2^{11}$

2. No single corner twist should be present =  $3^8/3 = 3^7$

3. No two edge pieces should be swapped =  $8!/2$

- That is,

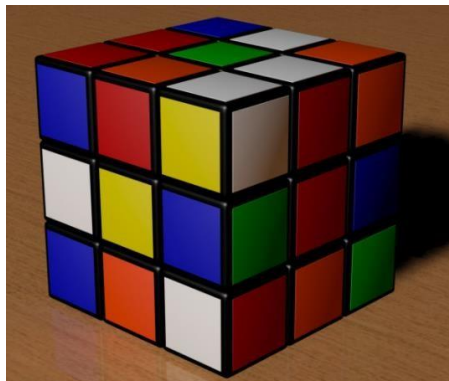
$$12! * 8! * (2^{12}) * (3^7) / (2 * 2 * 3) =$$

- The total number of possible solutions on a Rubik's cube are =

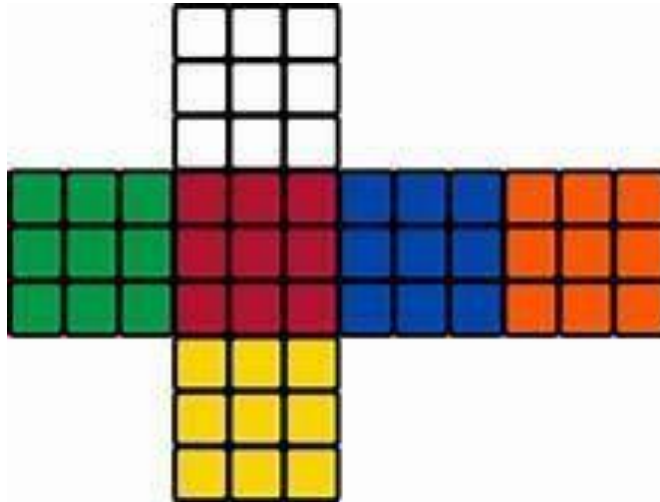
$$43,252,003,274,489,856,000$$

# Implementation of Hill cipher with Rubik's Cube

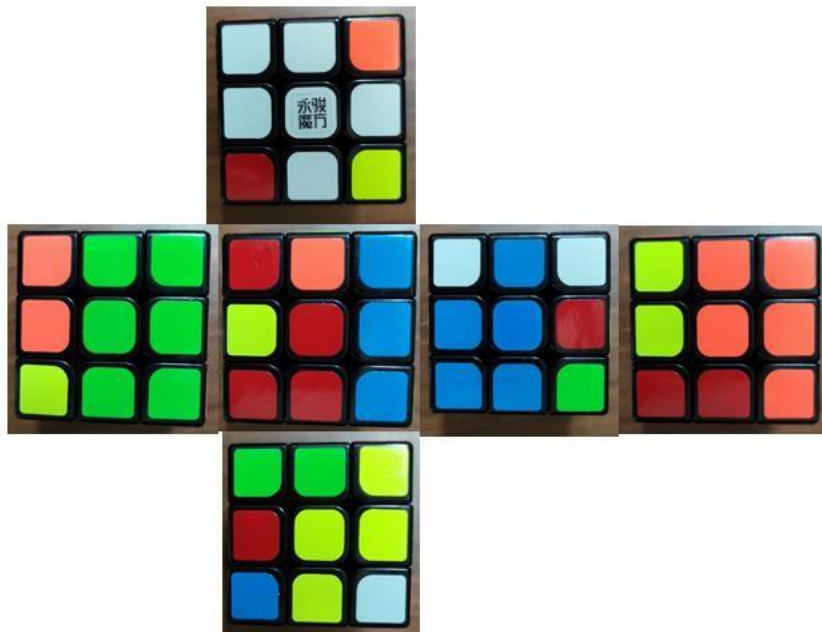
- Suppose consider the Rubik's cube is partially solved (Ex: White cross). Every time when we flip an edge or twist a corner, we get different corner and edge pieces. Initially, the cube is in scrambled position or whatever input we give.
- For example:



- Unfolding the cube and making into 2D will appear like as shown:



- After solving up to white cross, let us assume the six faces of cubes are shown like below:

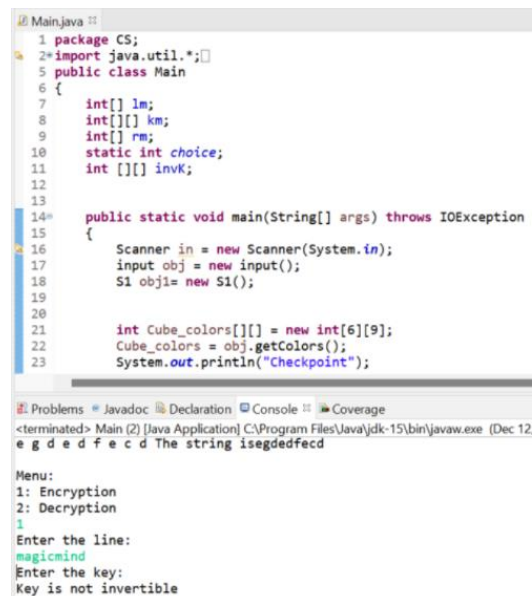


- Let's take a random side (say, green). And also considering the colors as numbers ranging from 1 to 6.
- Let,
  - 1 = GREEN, 2 = WHITE, 3 = BLUE, 4 = YELLOW, 5 = ORANGE, 6 = RED
 If we apply algorithm on green side, it takes input as:

1	1	1
1	1	1
5	5	4

- First thing necessary is to find determinant of the matrix. The determinant of matrix is 0 because two columns are same. Therefore, the matrix is not invertible and cannot act as a Key matrix for both encryption and decryption.

Let's take a look how eclipse works on this:



```

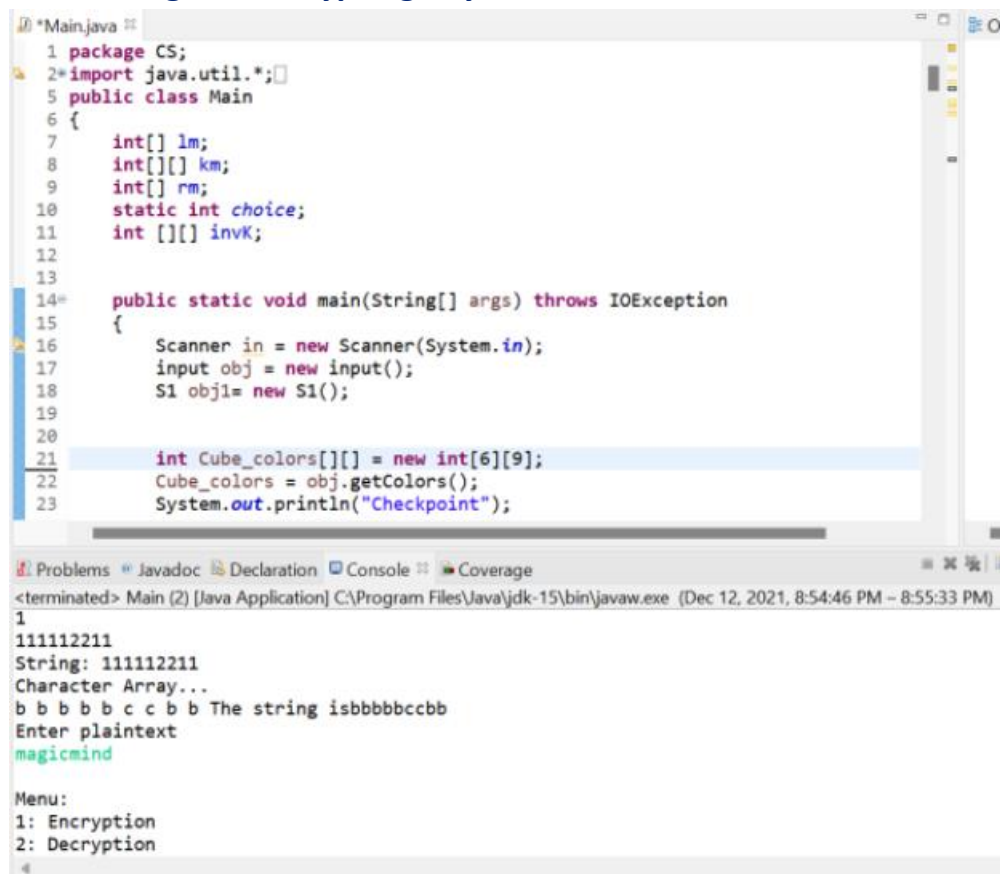
Main.java
1 package CS;
2 import java.util.*;
3 public class Main
4 {
5     int[] lm;
6     int[][] km;
7     int[] rm;
8     static int choice;
9     int [][] invK;
10
11     public static void main(String[] args) throws IOException
12     {
13         Scanner in = new Scanner(System.in);
14         input obj = new input();
15         S1 obj1= new S1();
16
17         int Cube_colors[][] = new int[6][9];
18         Cube_colors = obj1.getColors();
19         System.out.println("Checkpoint");
20
21     }
22 }
23
Problems Javadoc Declaration Console Coverage
<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (Dec 12, 2020)
e g d e d f e c d The string is egdedfecd

Menu:
1: Encryption
2: Decryption
1
Enter the line:
magicmind
Enter the key:
Key is not invertible
  
```

- Let us take a look how eclipse works when there is an invertible matrix with Rubik's cube combination.
- Assuming we have a matrix in the green side as:

1	1	1
1	1	2
2	1	1

- Now, IDE computes the determinant and executes the rest of code. Generating the encrypting key as follows:



```

1 package CS;
2 import java.util.*;
5 public class Main
6 {
7     int[] lm;
8     int[][] km;
9     int[] rm;
10    static int choice;
11    int [][] invK;
12
13
14    public static void main(String[] args) throws IOException
15    {
16        Scanner in = new Scanner(System.in);
17        input obj = new input();
18        S1 obj1= new S1();
19
20
21        int Cube_colors[][] = new int[6][9];
22        Cube_colors = obj.getColors();
23        System.out.println("Checkpoint");
    }
}

```

Problems Javadoc Declaration Console Coverage

<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (Dec 12, 2021, 8:54:46 PM – 8:55:33 PM)

1  
111112211  
String: 111112211  
Character Array...  
b b b b b c c b b The string isbbbbbccbb  
Enter plaintext  
magicmind  
Menu:  
1: Encryption  
2: Decryption

```

231         int m[][] = new int[N - 1][N - 1];
232         for (int i = 1; i < N; i++)
233         {
234             int j2 = 0;
235             for (int j = 0; j < N; j++)
236             {
237                 if (j == j1)
238                     continue;
239                 m[i - 1][j2] = A[i][j];
240                 j2++;
241             }
242             resultOfDet += Math.pow(-1.0, 1.0 + j1 + 1.0) * A[0][j1]
243                 * calDeterminant(m, N - 1);
244             break;
245         }
246         return resultOfDet;
247     }
248     public void cofact(int num[][], int f)
249     {
250         int b[][], fac[][];
251

```

Problems Javadoc Declaration Console Coverage

<terminated> Main (2) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (Dec 12, 2021, 8:54:46 PM – 8:55:)

Menu:  
1: Encryption  
2: Decryption  
1  
Enter the line:  
magicmind  
Enter the key:  
Result:  
syewieybg

- For example: If the key is not invertible, then the system gives us the choice to continue or exit. If we continue, we are going to encrypt/decrypt the message using playfair. The displayed results are shown, if the key for hill cipher is not invertible and we wish to continue encryption/decryption.

```
Character Array...
c c c c c c c c The string iscccccccc

Enter the plaintext/ciphertext:
magicmind
Enter the key:
Key is not invertible
Encryption of hill cipher is not possible
Would you like to continue Encryption
Enter your choice
1. Encryption of playfair
2. Decryption of playfair
1
Enter plaintext:
magicmind
Enter Key:
-----Key Matrix-----
  C   A   B   D   E
  F   G   H   I   K
  L   M   N   O   P
  Q   R   S   T   U
  V   W   X   Y   Z
-----
Encrypted text:
RG HK AL HO BY
```

### Procedure 1(Encryption):

- Step 1: Sender decides a plain text and generates a 54 bit key which is a Rubik's cube layout before solving partially. The range of a 54 bit key is taken as numbers in a range of 1-6 where, 1-6 represents different colors as mentioned above.
- Step 2: The 54 bit key is solved partially (white cross).
- Step 3: The algorithm takes the pieces randomly to generate hill cipher key.
- Step 4: Algorithm computes if the key has inverse or not, If there exists a inverse for 54 bit key, then Hill cipher can be successfully implemented.
- Step 5: if the key is invertible, it implements  $c = p * k \text{ mod } 26$  ( in small ASCII characters). Where cipher text is generated.
- Step 6: If the key is not invertible or doesn't form a square matrix, by default, The system asks whether to exit or proceed with playfair cipher encryption

### Procedure 2(Decryption):

- Step 1: Receiver needs to input exact 54 bit key(positions of cube).
- Step 2: The 54 bit key is solved partially (white cross) and the side which is needed to decrypt.
- Step 3: Algorithm implements  $p = c * k^{-1} \text{ mod } 26$  □ Step 4: Cipher text is reverted back.
- If Key is invertible Step 3 and Step 4 is implemented or else
- Step 3: We will take an input from user of ciphertext and using the obtained key. We decrypt the encrypted text to plain text.



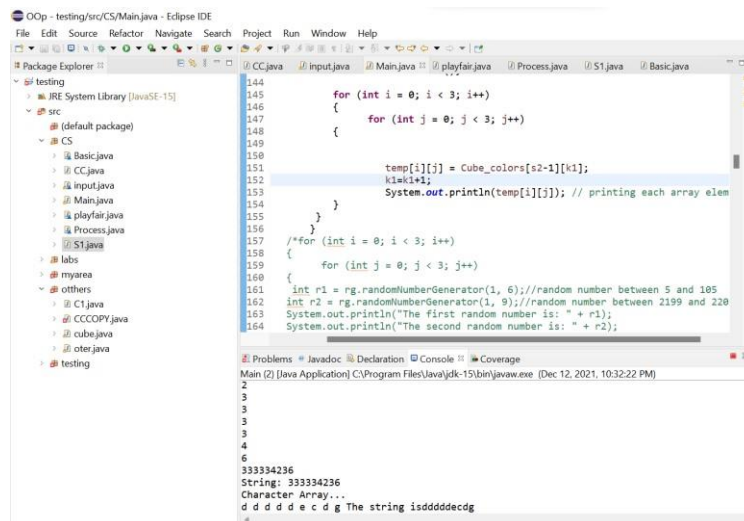
# Analysis:

- Each time if you give a different input of 1 bit in a key, the entire result is changed

For example:

Let us take the key set as:

111111554 225222624 332333162 114644341 644655555 333566646



The screenshot shows the Eclipse IDE with a Java project named 'testing'. The Package Explorer on the left shows the project structure. The main editor displays a Java file 'Main.java' with the following code:

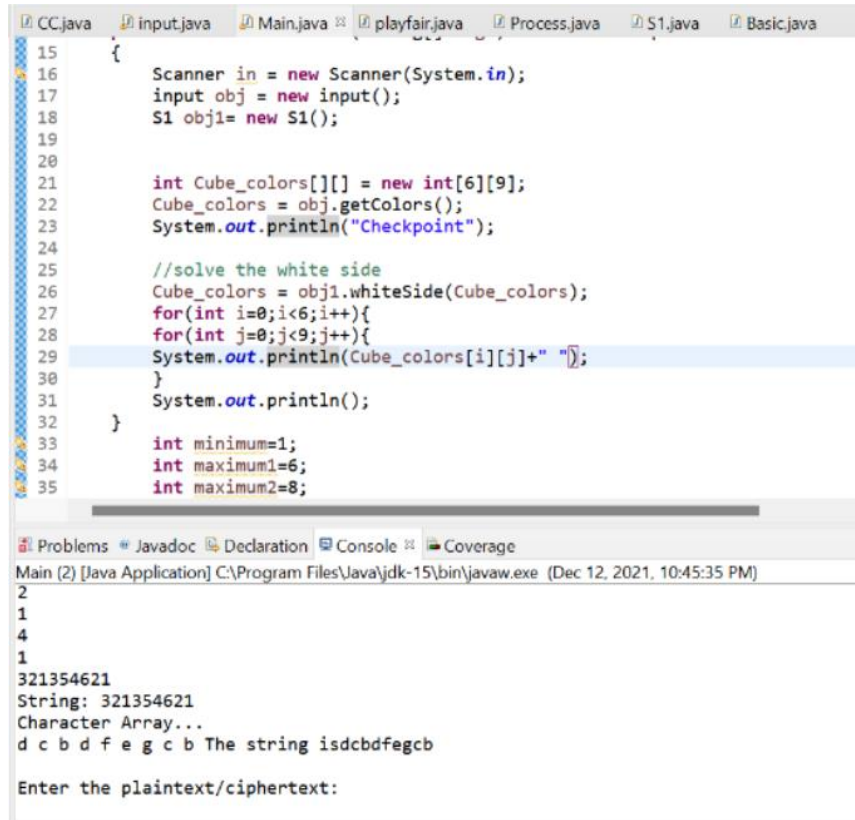
```
144     for (int i = 0; i < 3; i++)
145     {
146         for (int j = 0; j < 3; j++)
147         {
148
149
150             temp[i][j] = Cube_colors[s2-1][k1];
151             k1=k1+1;
152             System.out.println(temp[i][j]); // printing each array elem
153         }
154     }
155
156     /**for (int i = 0; i < 3; i++)
157     {
158         for (int j = 0; j < 3; j++)
159         {
160
161             int r1 = rg.randomNumberGenerator(1, 6); //random number between 5 and 105
162             int r2 = rg.randomNumberGenerator(1, 9); //random number between 2199 and 220
163             System.out.println("The first random number is: " + r1);
164             System.out.println("The second random number is: " + r2);
165         }
166     }
167 }
```

The console output at the bottom shows the following:

```
Main [2] [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (Dec 12, 2021, 10:32:22 PM)
2
3
3
3
3
6
333334236
String: 333334236
Character Array...
d d d d e c d g The string is d d d d e c d g
```

Now if we make a change:

211111554 225222624 332333162 114644341 644655555 333566646



```

15 {
16     Scanner in = new Scanner(System.in);
17     input obj = new input();
18     S1 obj1= new S1();
19
20
21     int Cube_colors[][] = new int[6][9];
22     Cube_colors = obj.getColors();
23     System.out.println("Checkpoint");
24
25     //solve the white side
26     Cube_colors = obj1.whiteSide(Cube_colors);
27     for(int i=0;i<6;i++){
28         for(int j=0;j<9;j++){
29             System.out.println(Cube_colors[i][j]+" ");
30         }
31     }
32     System.out.println();
33
34     int minimum=1;
35     int maximum1=6;
36     int maximum2=8;

```

Problems Javadoc Declaration Console Coverage

Main (2) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (Dec 12, 2021, 10:45:35 PM)

```

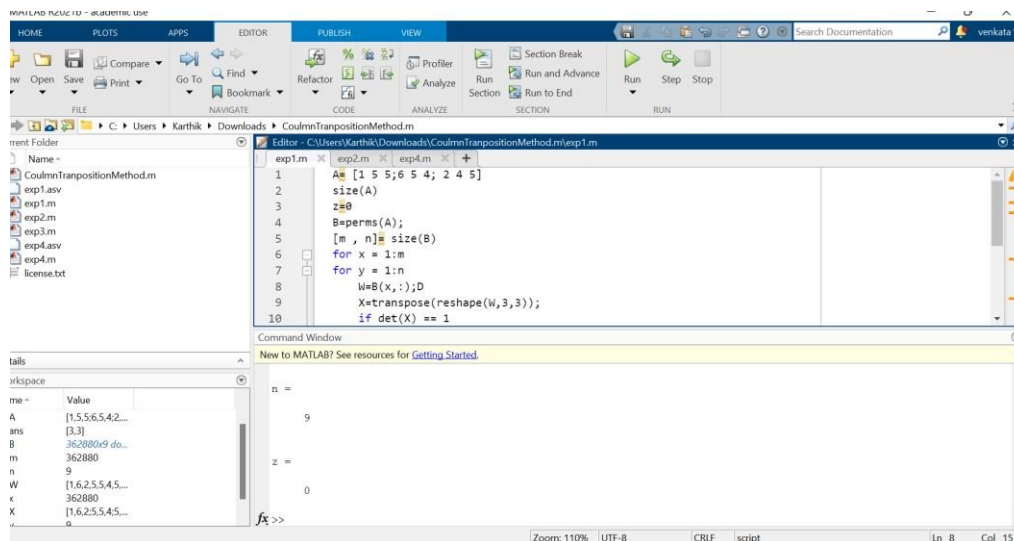
2
1
4
1
321354621
String: 321354621
Character Array...
d c b d f e g c b The string isdcdbdfegcb
Enter the plaintext/ciphertext:

```

- There are very few invertible matrices present in the range between 1 to 6, Rubik's cube further decreases its possibilities due to its cyclic properties.
- Using MATLAB, If we intend to permute the possibilities of a single set of hill cipher key, for each set possibility it is 0.02% or lesser that we find a invertible matrix.
- For example, let's take some random keys and design a sample MATLAB algorithm, Such that:

```
A= [1 5 5;6 5 4; 2 4 5]
size(A) z=0 B=perms(A);
[m , n]= size(B) for x
= 1:m for y = 1:n
W=B(x,:);
X=transpose(reshape(W,3,3));
if det(X) == 1 z=z+1 X end
end end
%number of invertible matrices in entire permutations
z
```

- The result is 0 out of Number of permutations possibilities which is 9!



The screenshot shows the MATLAB environment. The Editor window displays the script 'exp1.m' with the following code:

```
1 A= [1 5 5;6 5 4; 2 4 5]
2 size(A)
3 z=0
4 B=perms(A);
5 [m , n]= size(B)
6 for x = 1:m
7     for y = 1:n
8         W=B(x,:);
9         X=transpose(reshape(W,3,3));
10        if det(X) == 1
```

The Command Window shows the output of the script:

```
n =
    9

z =
    0
```

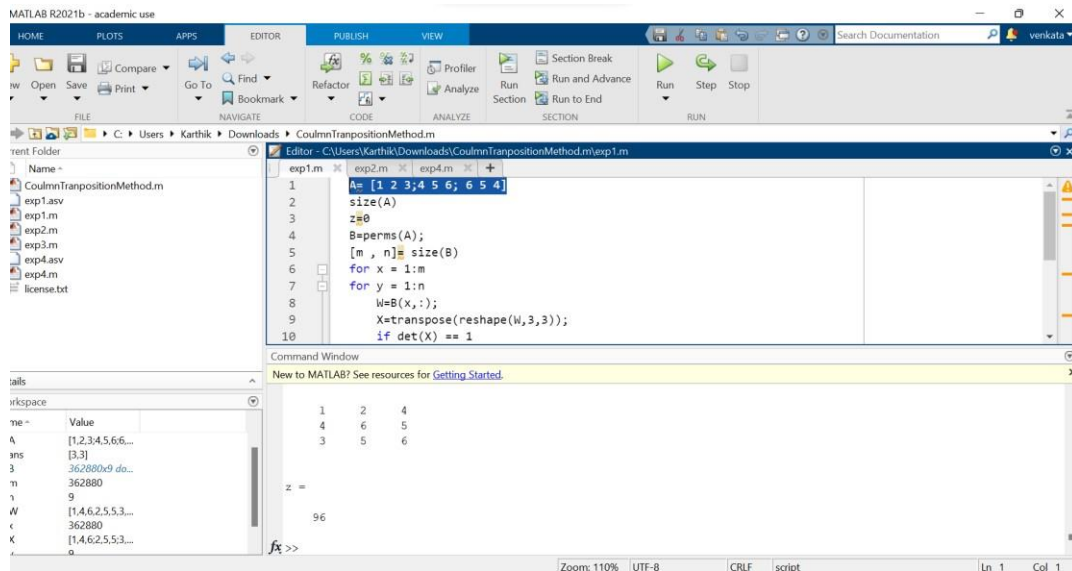
The Workspace window shows the following variables:

Variable	Value
A	[1 5 5; 6 5 4; 2 4 5]
ans	[3 3]
B	362880x9 double
m	362880
n	9
W	[1 6 2 5 5 4 5 ...]
X	[1 6 2 5 5 4 5 ...]
z	0

If we take another set, let's say:

$$A = [1 \ 2 \ 3; 4 \ 5 \ 6; 6 \ 5 \ 4].$$

The result is: 96 inverse matrices are present, if we permute the matrix A



```

1 A=[1 2 3;4 5 6; 6 5 4]
2 size(A)
3 z=0
4 B=perms(A);
5 [m,n]=size(B)
6 for x=1:m
7     for y=1:n
8         W=B(x,:);
9         X=transpose(reshape(W,3,3));
10        if det(X) == 1

```

Command Window

```

1      2      4
4      6      5
3      5      6

z =

96

```

- The average calculation is done for every set and found out, there are only 1, 50,000 (approx.) or even less inverse matrices possible for entire Rubik's cube.
- If we calculate the percentage of occurrence of an inverse in an Rubik's cube=  $(1, 50,000 / 4.32 \times 10^{19}) \times 100 = 3.487 \times 10^{-13}$ , which is impossible task to get the encrypted text back to original text unless the key is invertible.

## Benefits of Project:

- To utilize the maximum number of solutions of a simple Rubik's Cube in the symmetric key encryption.
- Implementation of cycle characteristics on an algorithm.
- Statistical analytic attack can be minimized.
- The constraint of finding an invertible matrix is minimized by Playfair cipher for encryption/decryption which if it is necessary.

## Constraints of Project:

- The number of actual solutions for Rubik's Cube decreases. (less than 43 quintillion possibilities, depends on number of fixed pieces (white cross))
- Hill cipher may not have invertible matrix of 1 in 1 million. Therefore decryption of the encrypted message is not possible
- The 54 bit key should result in an invertible key space and also while decryption the exact 54 bit key should

## Acknowledgment:

- <https://www.youtube.com/watch?v=ZtlMkzix7Bw&t=1s> (Inspired implementation of Rubik's cube algorithm)
- <https://www.researchgate.net/publication/328693056> Application of Hill Cipher Algorithm in Securing Text Messages
- <https://www.researchgate.net/publication/357177075> An implementation of Hill cipher and 3x3x3 rubik's cube to enhance communication security
- <https://www.researchgate.net/publication/286458327> Design and Analysis of Playfair Ciphers with Different Matrix Sizes
- <https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-net.2017.0196>
- <https://cjme.springeropen.com/articles/10.1186/s10033-018-0269-7> (Rubik's Cube mechanism)