Encryption Using the Rubik's Cube

BY

Weiqi Feng

A Study

Presented to the Faculty

of

Wheaton College

in Partial Fulfillment of the Requirements

for

Graduation with Departmental Honors

in Mathematics

Norton, Massachusetts

May 2019

Abstract

Rubik's Cubes, estimated to be the world's besting-selling toy, is a fascinating puzzle that I believe most people have attempted to solve. However, as anyone who has tried knows, Rubik's Cubes are hard to solve. Cryptography shares the essence with playing Rubik's Cubes; to fix the scrambled pattern. In this thesis, we discuss the advantages and challenges of using the Rubik's Cube as the basis of an encryption system.

We also test the Rubik's Cube encryption under the formal definition of security while seek additional opportunities provided by using 4 by 4 by 4 and larger cubes. Finally we construct a key-exchange protocol based on the Rubik's Cube using a structure similar to Diffie-Hellman key exchange.

Acknowledgments

I want to give a special thanks and my sincerest appreciation to my thesis advisor Professor Tommy Ratliff, who made this work possible. His insightful guidance and expert advice have been invaluable throughout all the stages of my work. He consistently allowed my creativity and independence to be shown in this thesis but steered me in the right direction whenever I needed it.

Then I want to express my gratitude to my committee member Professor Rachelle DeCoste and Professor Martin Gagné. I also want to thank my dearest friend Fiona Xu and Heydrick Padilla for their endless support and accompany.

Finally, I must express my very profound gratitude to my parents and my grandparents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Contents

Chapter 1

Introduction

Ever since I was a child, I liked playing with Rubik's Cubes. I was told by my uncle that there is a lot of mathematics behind this fascinating toy, such as the Rubik's Cube can be viewed as a group. Now, I am at the age where I gathered enough skills to understand the complexities of the Rubik's Cube. In this thesis, I investigate how to link Rubik's Cube with cryptography, a subject I obsessed with since sophomore year of college.

We start with discussing what cryptography is and listing motivations for us to integrate Rubik's Cubes with encryption protocols. While presenting design ideas of the Rubik's Cube encryption, we consider limitations and make corresponding improvements. Then, under the formal definition of security, we test if our Rubik's Cube encryption is well designed, and tweak it to make the encryption protocol better under the definition of security. Finally, we illustrate some known structures of the Rubik's Cube groups and introduce a Rubik's Cube key exchange protocol which has a similar structure that the Diffie-Hellman key exchange protocol uses.

1.1 What is cryptography

The definition of cryptography[3] given in the Merriam-Webster dictionary is "*secret writing*". This is historically precise, since in ancient times, people applied the idea of cryptography solely to enable secret communication. Most ancient cryptography schemes, as well as some modern ones look like some kind of puzzle. Yet creating and breaking the scheme would rely on how well you can manipulate the puzzle.

One basic scheme is the Substitution cipher, which replaces each unit of the plaintext to form the ciphertext. If we are working with English, then each unit of plaintext will be a letter. One trivial example could be substituting each letter

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

Figure 1.1: Map letter to letter

to another unique letter as displayed in Figure 1.1. If we encrypt the sentence *"life is like a box of chocolates"* with the relations specified in Figure 1.1, we get *"ifsafpifhazelwlsbdlblizqap"* back as the ciphertext. Correspondences between plaintext and ciphertext can be very creative as long as the communicating parties hold the same information. The "creative" example is shown in Figure 1.2, where we are replacing

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
✱	🏛	📦	💎	🖂	🎞	⚙	#	🖻	🖼	🔑	🍃	🧲	☰	🕹	🚀	🔳	🔀	🖫	🏷	👤	📹	📶	🌿	📺	🖵

Figure 1.2: Map letter to icon

letters by icons which are completely irrelevant. We can encrypt the same sentence " *life is like a box of chocolates"* again and this time we get "🖻🎞🖫✱🎞🚀🖻🎞 #✱🖵🖂🍃📶🍃🖫🏛💎🍃🏛🍃🖻🖵🔳✱🚀" back.

Intuitively, we know the security of substitution cipher is based on the fact that the correspondence between plaintext and ciphertext is unknown to the attacker. Although we may find the second substitution to be more abstract, the brute force attack or frequency analysis will be equally effective on both.

Historically, the biggest motivation for people to research in this field, and most applications, were military related. Thus cryptography seemed far away from most people's daily life. However, cryptography has primarily changed since the invention of computers and the internet. It has merged deeply into our everyday life though you may not have noticed. Whenever you log in your email account or when you purchase an item from an online shop, you have undoubtedly used cryptography. While you are communicating with the web servers, your email account password or credit card number will first be encrypted and then sent through the networks. If that critical information were never encrypted, anyone who can monitor your network would be able to retrieve your private information.

While secure communication is still the major motivation of cryptography, the study of cryptography has developed other related branches that are equally important. Some of the noticeable branches are key exchanging protocols, message authentication code, hash functions and more. Later in this writing, we will discover more about key exchanging protocols as they are crucial components of symmetric encryptions. Message authentication code, usually shorten as MAC, is used to confirm that the message came from the stated sender and has not been modified. Hash function is one commonly used tool to build valid MACs.

Computers have also brought us ample computational powers. In modern times, a 30 dollars Raspberry Pi can perform an exhaustive attack on the substitution cipher on 26 English letters showed in our first example in the time it takes to make a cup of

coffee. Thus, deviating from studying the art of clever puzzles, modern cryptography makes extensive use of mathematics, including fields such as abstract algebra, number theory, statistics, combinatorics, and computational complexities.

In a nutshell, cryptography has gone from the set of clever puzzles concerned with ensuring secret communication for the military to science that provides security for ordinary people across the globe. Modern cryptography heavily relies on mathematical theory and computer science practice. Cryptographic protocols are designed around some computational hardness assumptions, meaning that theoretically, it is possible for one to break such protocol, but it is infeasible to do so by any practical means.

1.2 Rubik's Cube and cryptography

Rubik's Cube, as shown in Figure 1.3, is a three dimensional puzzle invented in 1974 by Hungarian professor Erno Rubik. The most common Rubik's Cube has six faces. The faces of the cube are covered by the following solid colors: white, red, blue, yellow, orange and green. For a solved cube, the white face is always opposite to the yellow face, red is opposite to orange and blue is opposite to green. As shown in Figure 1.3, the Rubik's Cube has three small cubes on each side. We say that this is a 3 by 3 by 3 cube, or that it is a cube with side length of 3. Clearly each face of the cube includes nine squares. In future discussions, we will call those squares *cubies*.

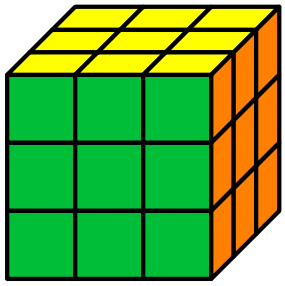

Figure 1.3: Rubik's Cube

After years of developing, Rubik's Cubes now exist in many forms. For the six faced cubes, their side lengths may vary from 2 to 13 or even larger number. However,

since their structures are similar, they share a lot of properties in common. For example, regardless of the side length, all six faced cubes can be solved in an almost identical fashion. We will refer Rubik's Cube with side length n as C_n. For instance,

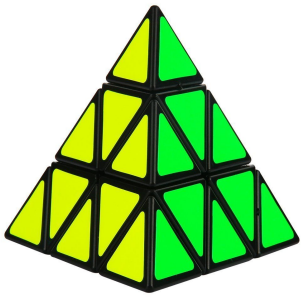

Figure 1.4: A solved pyraminx cube

Figure 1.5: A solved pentagon cube

the normal 3 by 3 by 3 Rubik's Cube will be denoted as C_3. Also when n is an odd number we will call C_n an odd cube and when n is an even number we will call C_n an even cube. Further more, Rubik's Cubes also have other structures; "cubes" with four or twelve faces as listed in Figure 1.4 and Figure 1.5 have also been made in production.

Why would we relate Rubik's Cubes and cryptography? As we mentioned in the last section, cryptographic protocols are designed around some hard problems, and Rubik's Cubes are known to be hard to solve. One may argue that this is not true since the best human players can solve a 3 by 3 by 3 cube in about 6 seconds. By carefully following the developed algorithms, anyone can solve a well shuffled Rubik's Cube within a couple of minutes.

Suppose we play the Rubik's Cube differently; after shuffling the cube, we give it to a player without letting him/her observe it. Therefore, to solve the puzzle, the player has to guess and shuffle it randomly. Then, the player needs to confirm with a judge to see if the cube is solved. The player wins if the cube is solved, but if the

cube is not solved, it will be reversed back to its beginning state and the player has to guess again. Under this setting, the Rubik's Cube will become much harder to solve because the player has to go through an enormous amount of possible cases.

This is essentially what we will do in Chapter 2. We place our plaintext on the cube and remove the colors from the faces and then shuffle the cube. Let us consider each different shuffling result of the Rubik's Cube as a state. The most common cube C_3 has over 43 quintillion different states. It will take decades for a powerful desktop to run through all of them. Hence it will almost be impossible for a human player to run through all possibilities and guess if the cube is correctly solved. Also, when the side length of a Rubik's Cube grows, the number of states the cube produces will grow significantly fast. Being such a powerful device, Rubik's Cubes can be used to design schemes that are safe against exhaustive search attacks. In addition, we can algebraically view Rubik's Cubes as groups. The nice properties of groups are widely used in cryptography, and we will see how to use the group structure of Rubik's Cube to define a key exchange protocol.

1.3 Preliminaries

First let's take a look at ***Kerckhoffs's principle***:

> *The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without affecting the security of the cipher method.*

This principle tells us that the security of an encryption protocol does not rely on the encryption protocol procedures being unknown. One may wonder how do we obtain

the security while the eavesdropper knows all the details. But there is one thing the eavesdropper does not know, which is the ***key***. In fact, the security of all classical encryption schemes depends on the key shared by the communicating parties being unknown by the eavesdropper. For our substitution cipher example, we consider the correspondence between plaintext and ciphertext as the key. Like we said, the security of substitution cipher depends on the key being secret. This scenario is known as the *private-key* setting.

Formally, to create a private-key encryption scheme, we need to specify a message space $\mathcal{M}$. This message space $\mathcal{M}$ defines the set of "legal" messages. Taking the substitution cipher described in Figure 1.1 as an example, a "legal" message could be any combination of English letters without spaces or punctuations, regardless of if it has any actual meaning. In contrast, any message containing a Greek letter will make the message "illegal". Along with defining the message space, we need three algorithms for a private-key encryption scheme, which are **Gen**, **Enc** and **Dec**:

- **Gen** is the procedure for generating keys. It is a probabilistic algorithm that outputs a key k with desired length according to some distribution.

- **Enc** is the procedure for encrypting the message. It takes a key k and a message $m \in \mathcal{M}$ as inputs. Then it uses k to encrypt m and outputs ciphertext c. We denote this process by $\mathbf{Enc}_k(m)$.

- **Dec** is the procedure for decrypting the message. It takes a key k and a ciphertext c as inputs. Then it uses k to decrypt c and outputs plaintext m. We denote this process by $\mathbf{Dec}_k(c)$.

It is worth noting that the set of all possible keys output by the key-generation

algorithm is called the key space and is denoted by $\mathcal{K}$. In most cases, **Gen** randomly picks one key from the key space uniformly.

With these algorithms, we can move on to construct the set of legal encryption schemes. One requirement an encryption scheme must satisfy is defined as the following.

Definition 1.1. ***The correctness of an encryption scheme***
For any encryption scheme $\Pi = (\boldsymbol{Enc}, \boldsymbol{Gen}, \boldsymbol{Dec})$ *and for every* k *generated by* ***Gen*** *and every message* $m \in \mathcal{M}$*, it must hold that*

$$\boldsymbol{Dec}_k(\boldsymbol{Enc}_k(m)) = m$$

This simply means that we accurately decrypt any encrypted message when we hold the correct key. To provide an example of a valid encryption scheme, we simply define $\Pi = (\mathbf{Enc}, \mathbf{Gen}, \mathbf{Dec})$, where $\mathbf{Enc}_k(m) = m$ and $\mathbf{Dec}_k(c) = c$. In words, both **Enc** and **Dec** do not modify the input string regardless of what the key k is. So Π is a valid encryption scheme as it satisfies the correctness of an encryption scheme. But obviously, Π is not a secure scheme. As we explore more about using Rubik's Cube to perform encryption, we will also explore what it means for an encryption scheme to be secure.

Chapter 2

Encryption

In this chapter, we introduce notations for fundamental moves of the most common Rubik's Cube C_3. We then work on constructing a group using states of C_3. We explain how the encryption protocol is built using the Rubik's Cube and move on to analyze the possible attacks against this encryption protocol. Finally, we layout some improvements to the encryption protocol.

2.1 The classic Rubik's Cube

Recall Figure 1.3, C_3 has 3 layers where each layer is formed by 9 smaller cubes, so there are $3 \cdot 9 = 27$ cubes in total. Among those 27 small cubes, 26 small cubes are visible since the center cube is surrounded by them and thus, hidden. If you take apart one Rubik's Cube, you will find that the center cube does not actually exist. In place of the center cube, there is a special structure that holds all other pieces together.

We can split the 26 small cubes into three different types. As shown in Figure 2.1, the small cubes in the corners are called "corner cubes", which have three visible

colored pieces and there are 8 of them in total. The small cubes lie in the middle of each edge are called "edge cubes", which have two visible colored pieces and there are 12 of them. Finally, the cubes with a single visible colored piece located at center of each side of the cube are called "center cubes". Same as the number of sides a cube has, there are 6 of them.

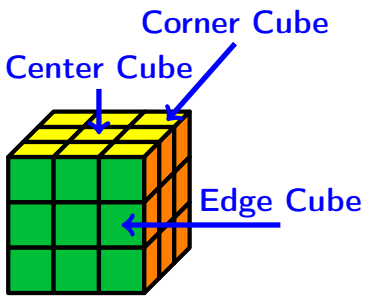

Figure 2.1: Small cube types

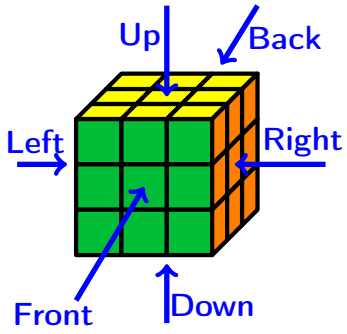

Figure 2.2: Rubik's Cube Notation

In future discussions, we will refer sides of the Rubik's Cube as faces. To distinguish the six faces, we will call them right face, left face, up face, down face, front face and back face as shown in Figure 2.2. Following this naming system, we can assign notations to the moves of C_3. The most basic move one can do it to rotate a single face of the cube. We use U to denote a clockwise rotation of the up face. That is looking at the up face, turn it 90° clockwise. Similarly, we use letters R, L, D, F and B to denote the clockwise 90° rotation of the corresponding faces. For each of these six moves, there are three possible angles of rotation, which are 90° clockwise, 180° clockwise and 270° clockwise. In total there are $6 \cdot 3 = 18$ possible moves and we will call these the *fundamental moves*.

To represent the clockwise 180° rotation of the corresponding faces, we simply add a number 2 after the capital letters. Therefore, $U2$ represents the move where we

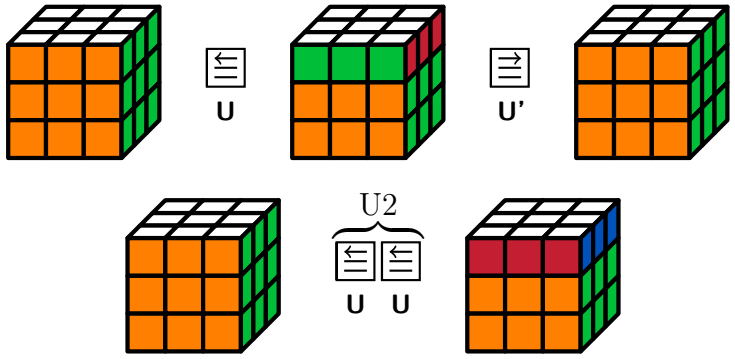

Figure 2.3: Rotation example

face the up face and rotate it 180° clockwise. For the 270° clockwise rotation, which is same as a 90° counterclockwise rotation, we add an apostrophe after the capital letter to denote it. Then U' represents the move where we face the up face and rotate it 90° counterclockwise. All above notations defined are illustrated in Figure 2.3.

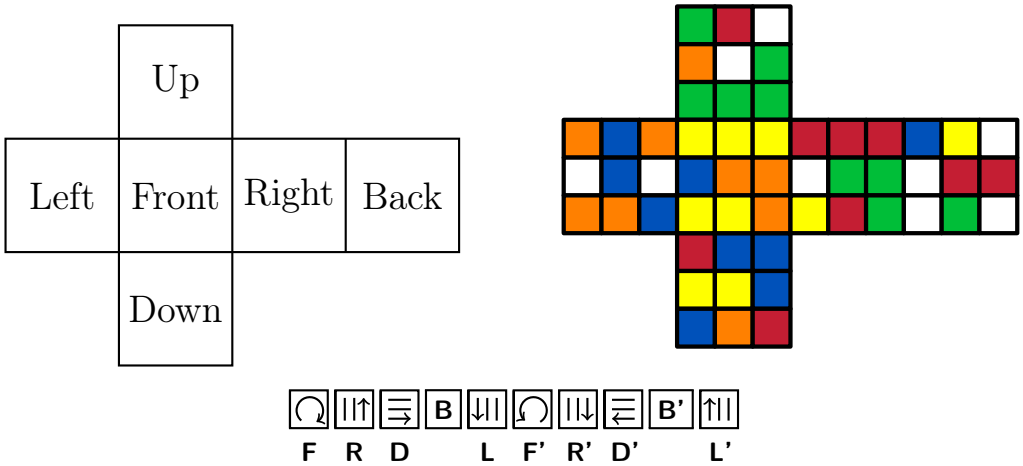

Figure 2.4: A state of the Rubik's Cube

As we mentioned earlier, we refer each possible shuffling result of the cube as a state. Figure 2.4 is one possible state of the cube. The shuffling can be any combination of the fundamental moves applied to a solved cube. Each state of the cube is unique, though infinitely many different shufflings could reach it. Suppose we

are at a state S that can be reached by series of fundamental moves M. Then if we add "$R\,R\,R\,R$" at the end of M, we are still at state S since "$R\,R\,R\,R$" simply rotates the right face 360° clockwise and does not actually change it. In fact, we can insert any number of "$R\,R\,R\,R$" anywhere in M. Also, we do not have to always use R, any four fundamental moves will rotate one face of the cube by a multiple of 360° and hence do not modify the cube. Thus we have infinitely many ways to reach S.

We now want to show that we can make the set of states of C_3 into a group. Let us denote the group as G_3 where the group operation is $*$, and each element of G_3 is a state. Though arguments below correspond to C_3, we can use them to construct group G_n for states of an arbitrary cube C_n in a similar fashion.

Assume S_1 and S_2 both represent a state of the classic 3 by 3 by 3 cube, that is $S_1, S_2 \in G_3$. Assume S_1 can be reached by a series of moves M_1 applied to the solved cube and similarly S_2 can be reached by a series of moves M_2 applied to the solved cube. Then $S_1 * S_2$ is the state you could reach to by first applying M_1 and then M_2 to the solved cube, or equivalently, you can directly apply M_2 to S_1. Then it is obvious that if we pick a different M' which takes us from the solved cube to S_1, $S_1 * S_2$ still gives us the same result. Therefore the operation $*$ is well defined. We now can show a simple proof that G_3 is indeed a group under $*$.

Claim: G_3 is a group under $*$.

Proof:

- Show closure.

  Suppose we have $S_1, S_2 \in G_3$. From above description, we see that $S_1 * S_2$ can be reached by combining two series of moves. Thus $S_1 * S_2$ is indeed a state of the cube. Therefore G_3 is closed.

- Show identity.

  The solved cube is the identity and we denote it as e. It represents the "empty" move. Suppose we have $S \in G_3$, then $e * S$ means do nothing and move to S. Similarly $S * e$ means applying nothing to S. Therefore the solved cube is the identity.

- Show associativity.

  Suppose we have $S_1, S_2, S_3 \in G_3$. Clearly $(S_1 * S_2) * S_3 = S_1 * (S_2 * S_3)$ since the order we apply those moves does not change even if we combine two states together.

- Show inverse.

  Suppose we have $S \in G_3$ and S can be reached by a series of moves M applied to the solved cube. Each move in M is one of the 18 possible fundamental moves. All fundamental moves have their inverses. For example, U can be reversed by U' and $U2$ can be reversed by $U2$. Thus there must exist a series of moves M' that reverses M. We can find M' by inverting of all movements in M in reverse order. That is, the move that was done the last shall be undone first. Applying M' to the solved cube to obtain S' then $S * S' = S' * S = e$.

Therefore $(G_3, *)$ is a group. □

2.2 The design principle

"Do not worry about your difficulties in Mathematics. I can assure you mine are still greater." This is one of my favorite quote from Albert Einstein. Let us assume that we now want to secretly pass this quote to someone using the Rubik's Cube. The general

idea is, as we mentioned in Chapter 1, instead of letting each cubie to hold a color, we use cubies to hold letters. We then need to agree on an order that we put the letters in. As long as we are consistent, we can order the six faces in any way we desire. Once we fill all the faces, we can shuffle the cube to obtain the ciphertext. But before we get started on filling the cube with letters, some pre-processing to the plaintext is essential. We want to remove blanks and punctuations, since their existences in the ciphertext leaks important information about the plaintext. An eavesdropper can count the number of blanks to deduce how many words were encrypted. Afterwards, we also want to make sure all the letters are either capitalized or lower cased. Here we choose to capitalize everything and the processed string looks like the following:

```
DONOTWORRYABOUTYOURDIFFICULTIESINMATHEMATICSI
CANASSUREYOUMINEARESTILLGREATERALBERTEINSTEIN
```

Since we are using the classic Rubik's Cube C_3, we can put $3 \cdot 3 = 9$ letters on each face. At one time, we can use the cube to encrypt at most $9 \cdot 6 = 54$ letters. Our message is too long to fit, but we can take 54 letters and encrypt them first. Here are the first 54 letters split into chunk of 9:

```
DONOTWORR YABOUTYOU RDIFFICUL TIESINMAT HEMATICSI CANASSURE
```

Let us agree on the order of filling the letters as the following: up face $\rightarrow$ front face $\rightarrow$ right face $\rightarrow$ down face $\rightarrow$ back face $\rightarrow$ left face. The cube with letters filled is displayed in Figure 2.5. We then need to determine a sequence of fundamental moves as our encryption key, which we denote as k. Suppose we apply "$R\,U\,B2\,L'\,D2\,F\,R'\,D\,B\,L2$" to the cube. The shuffled cube is shown as Figure 2.6. To obtain the entire ciphertext c, which is shown below, we simply read out all the letters the same order we put them in.

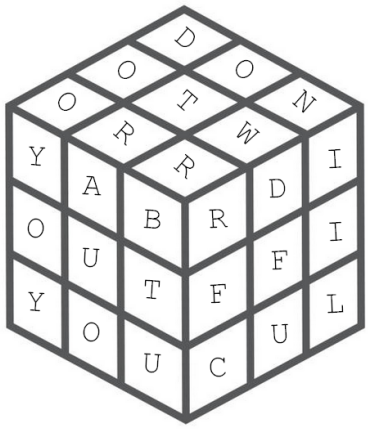

Figure 2.5: Cube with plaintext

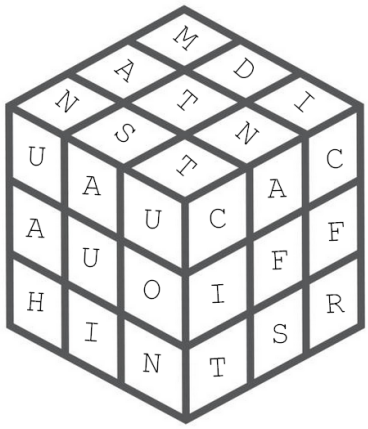

Figure 2.6: Cube with ciphertext

MDIATNNST UAUAUOHIN CACIFFTSR YWLTIEBRM EOCOTROID ESIOSUYAR

We now send the ciphertext c to the receiver. It is crucial for us to have shared the key secretly with the receiver. We will discuss a method to do so in Chapter 5. Upon receiving the ciphertext, the receiver can decrypt the ciphertext by first finding the inverse of the key, which we denote as k'. By following the step described in proving the inverses of group G_3, we find k' is "$L2\,B'\,D'\,R\,F'\,D2\,L\,B2\,U'\,R'$". Then the receiver can fill the cube with the ciphertext and apply k' to the cube.

Let us take a closer look at what happened during the encryption. We notice that all letters that were in plaintext are present in the ciphertext. However most of them ended up in different locations. If we observe the center of each face, we can find that they all remained in the same places. This is not a coincidence since none of the basic moves changes the center. Therefore among the 54 letters, at most 48 letters can be permuted.

2.3 Brute force attacks

Let us consider the brute force attacks against this protocol. There are two relative obvious ways of doing so. From above discussion, we know the ciphertext is just one possible permutation result of 48 letters within the plaintext. Thus if we list out all elements of S_{48}, the permutation group on 48 elements, and apply each to the ciphertext, one of the permutations must revert the ciphertext back to the plaintext.

Though the group S_{48} has $48! \approx 1.24 \cdot 10^{61}$ elements, later in this thesis, we will show that not all those permutations are needed to be checked. Even though Rubik's Cube permutes 48 elements, the size of the group G_3 is smaller than S_{48}, since there are invalid states that cannot be reached by series of fundamental moves. Potentially you can take the cube apart and put it back together to get to a invalid state. We will talk about what states are invalid and why they are invalid in Chapter 4 when we explore more about the group structure.

The other possible attack is to search the inverse of the encryption key. Without knowing the length of the key we used, the attacker can start with trying out all combination of fundamental moves with length of 1, length of 2 and so on. For each combination the attacker selects, he/she applies it to the cube and observe if the result makes sense. As an example, there are 18^2 possible keys with length of 2 since every time the attacker can pick one from the 18 fundamental moves. It follows that, in our example, the number of keys the attacker needs to go through is upper bounded by $\sum_{i=1}^{10} 18^i \approx 3.78 \cdot 10^{12}$.

By comparing the two brute force attacks described above, we notice that finding the inverse of the key would be easier for the attacker. It seems that there is a straightforward approach to make this brute force attack infeasible, which is to

increase the key length. Instead of using a key with a length of 10 fundamental moves, we can randomly pick a key that is formed by 1000 fundamental moves. Then if we follow above methods, we will find the number of keys the attacker have go through being bounded by $\sum_{i=1}^{1000} 18^i \approx 1.98 \cdot 10^{199}$ which is obviously a way larger number. However this approach will not work as expected due the existence of the God Number.

2.4 The God Number

We know that mathematicians love the Rubik's Cube as they are amazed by how such a seemingly regularly puzzle can hold so many secrets. Ever since the problem is invented, perhaps the biggest mystery of all is the upper bound for number of fundamental moves to reach to all possible states. The lower bound is proved earlier in 1995 by Michael Reid[8]. Reid found that the superflip, which keeps all small cubes in their solved locations but flips the orientation of all edge cubes, as displayed in Figure 2.7 requires at least 20 fundamental moves. It took mathematicians about 30

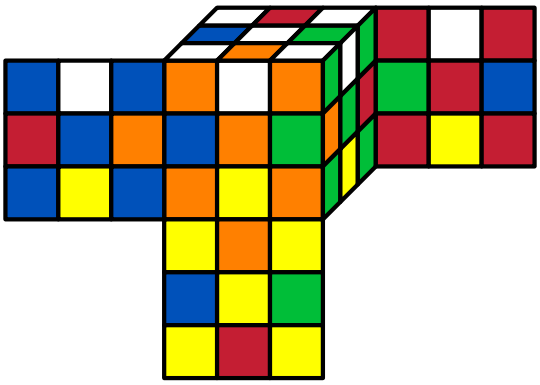

Figure 2.7: Superflip

years to prove that this is an upper bond. We formally define the God Number as:

Definition 2.1. ***The God Number***

Let $m(S_1, S_2)$ denote the minimum number of fundamental moves to transform C_3 from S_1 to S_2. Then the God Number is $max\{m(S_1, S_2) \mid S_1, S_2 \in G_3\}$.

With about 35 CPU-years of idle computer time donated by Google, a team of researchers essentially solved every position of the Rubik's Cube and showed in 2010 that the God Number is 20[8]. Another way to express the meaning of the God Number is that any series of fundamental moves with length longer than 20 can be reduced to a series of fundamental moves with length less or equal to 20. Thus, we can not find two states in the Rubik's Cube that are more than 20 fundamental moves away. We can also say that the Rubik's Cube group G_3 we defined has a diameter of 20 fundamental moves.

Therefore, even if we have chosen a key with length of a thousand fundamental moves, the attacker will find the inverse of the key with only searching for keys that having length of 20 fundamental moves or less. That is, the number of keys the attacker has to go through is really bounded by $\sum_{i=1}^{20} 18^i \approx 1.35 \cdot 10^{25}$, regardless to the key length we choose. It follows that the key search attack will almost always be easier than the attack on finding inverse of the permutation. It is also worthwhile to note that the God Number for larger cubes is unknown, so we are not able to perform a similar analysis on larger cubes.

2.5 Improvements

In this section, we discuss solutions that prevent the key from collapsing and other tweaks to our design to improve the security of our protocol in general. First, let us observe some additional features provided by Rubik's Cubes that we overlooked.

Suppose we fill a letter E in the right up corner cubie on the front face of the cube as shown in Figure 2.8. If we rotate the front face 90 degrees clockwise, as shown in

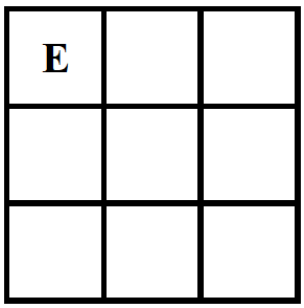

Figure 2.8: Front face with letter

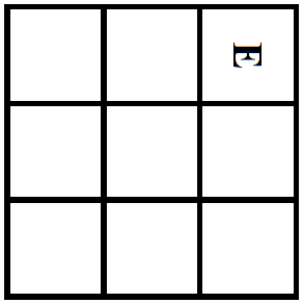

Figure 2.9: Front face with rotated letter

Figure 2.9, we can find the position of letter **E** changes. However, we also notice that the orientation of **E** changes. Since anyone who reads English can still recognize those letters even if their direction changed, we omitted this feature while developing the encryption scheme. To benefit from this feature, we can use something that rotates while the cube faces rotate. Therefore, instead of putting the letters, we can put four bits into each cubie, where each corner of the cubie holds one bit. We need to agree on an order of placing those bits, for example: left up corner → right up corner → right down corner → left down corner, so we get a complete cycle here. Assume that we are putting integers from 1 to 36 on one face of the cube in the order, the face will look like what is displayed in Figure 2.10. We now can run the above experiment again with input "1000." Following the order we agreed, we put the bits into the left up corner cubie on the front face as shown in Figure 2.11 and the face after the rotation is displayed in Figure 2.12. If we read out the bits the same order we put them in, we get "0100", which is different from the original input. Hence using bits helps us benefit from the face rotations of the Rubik's Cube. Though this method does not directly address the key collapsing issue, it gives us more flexibility to modify the encryption

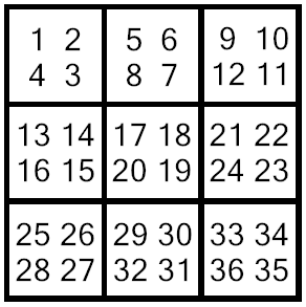

Figure 2.10: Bit ordering

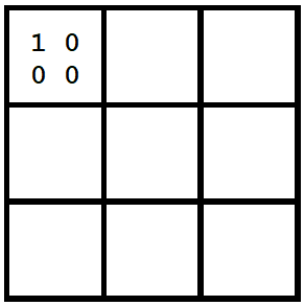

Figure 2.11: Front face with bits

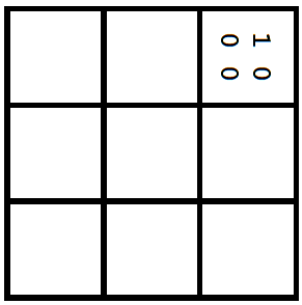

Figure 2.12: Front face with rotated bits

protocol. We will make further improvements based on this setting.

One obvious reason the key would collapse is because some fundamental moves commute with each other. Clearly, each fundamental move commutes with itself regardless of the angle; for example R, $R2$ and R' commute with each other. Also, the moves that rotate opposite sides commute with each other. For instance R, $R2$ and R' commute with L, $L2$ and L'. Suppose we find a series of moves within a random key of length 15 as "$R2\,L2\,R2\,L\,L$". Since those moves commute, they are equivalent to "$R2\,R2\,L2\,L\,L$". In words, what this chunk of key does is rotating the right face 360° clockwise and then rotating the left face 360° clockwise, which has precisely no effect on the cube. Thus we can merely remove this chunk from the random key and its length will collapse down from 15 to 10.

To resolve this issue, we can shift the content to the right by one bit each time

before we apply a move to the cube. Each one of the bits will move to the next position following the order they were put in. For example, recall Figure 2.10, the bit at location of number n will be shifted to the location of number $n+1$. Notably, the last bit on one face will become the first bit on the face next to it. For instance, the last bit on the up face will become the first bit on the front face. As shown in

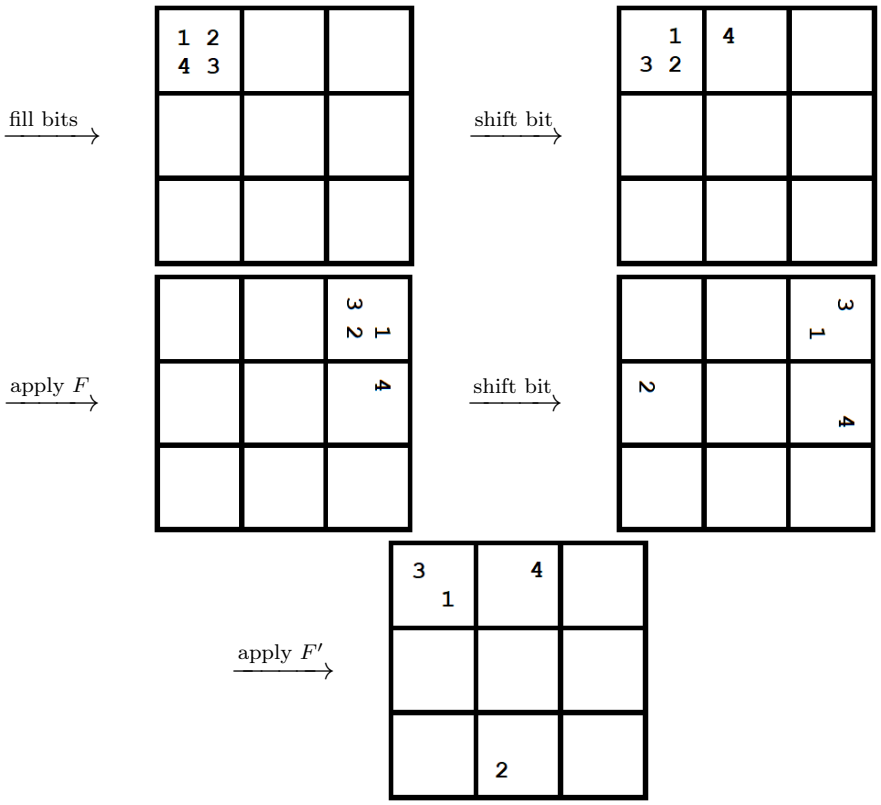

Figure 2.13: The bit shift method

Figure 2.13, we put "1234" in the right up corner of the front face and apply moves F and F' to the cube. We follow the procedures defined above and shift the bits before applying each move. Though "1234" is not a legal input to the Rubik's Cube encryption, we use it to clearly illustrate where each bits travels to. By observing the front face in the last row of Figure 2.13, we find the locations of all bits changed,

though F and F' cancel each other out. Among the four bits, two bits even moved to other cubies. Therefore, with the shift, the bits travel around even when moves commute and cancel each other out as fundamental moves. We believe this method effectively prevents the key from collapsing. Also, this method addresses the "fixed center" issue since the shift affects bits that are held by center cubies.

To examine the modified encryption protocol, we can try to encrypt my favorite quote from Einstein again with the modified Rubik's Cube encryption. Notice we first need to convert the English letters to their ASCII representations, so each letter will be 8 bits long. Since C_3 can hold 216 bits, we can encrypt $\frac{216}{8} = 27$ letters at once. The first 27 letters are "`DONOTWORRYABOUTYOURDIFFICUL`" and the following is its binary representation:

01000100010011110100111001001111010101000101011101001111010100100101001001
01011001010000010100001001001111010101010101010001011001010011110101010101
01010010010001000100100101000110010001100100100101000011010101010100110 0

We can use the same key "$R\,U\,B2\,L'\,D2\,F\,R'\,D\,B\,L2$" to encrypt the plaintext but this time we will account for the shift before applying each move to the cube. The encrypted ciphertext is:

00111010011010011011101001110000001010001110111111101001111110111101110 1
00100001100110001111000000100011111111011000001001010000000011010001000
01001110101001101001010011000101010010100000011010011000110001101010000 1

From Figure 2.10, we see that the center of the cube face goes from the 17th position to the 20th position. We thus can find, in the plaintext, the center of the up face holds 1001 and this value changed to 0111 after the encryption. Among the four bits,

128	Ç	144	É	160	á	176	░	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	│	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	█	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

Figure 2.14: Extended ASCII table

only one bit actually remained the same. So in this particular example, the shifting bits is effectively helping to change the bits held by the center cubes.

We can convert the ciphertext back to "letters". But since we can no longer guarantee that each chunk of 8 bits still represent a value that is less than or equal to 127, we need to rely on the extended ASCII table as shown in Figure 2.14 to convert the ciphertext back to "letters." The following characters are the results of the conversion: ":i‖p(∩θ√■!ÿ≡#²éP êNªö†J ÿ⊢í." As one can notice that while we are using the extended ASCII, many more special characters will be involved. We find that the only common letter that exists in both the ciphertext and the plaintext is the letter N, but the location of it has changed.

The essence of the shift on the bits is just a permutation. We described the shift as a minimum working example, but we are free to pick any permutation on the 216 bits as long as we can specify it. To add more security, we can select a more complex permutation and secretly share it as part of the key with other trusted parties.

2.6 Encrypt long messages

When we first defined the Rubik's Cube encryption, we noticed that we could not use it to encrypt the entire quote, since the Rubik's Cube can only fit a certain amount of the plaintext at once. In real life, we may often need to encrypt messages that are much longer. Thus, we introduce two methods that help us encrypt long messages.

The first method is designed targeting the Rubik's Cube encryption. We have been using C_3 for the encryption, but in Chapter 1 we mentioned that cubes with larger side length do exist. Hence we can use a Rubik's Cube with greater side length to extend the size of the message we are capable of encrypting. Suppose we instead use a 10 by 10 by 10 Rubik's Cube, denoting as C_{10}, we can encrypt $10^2 \cdot 6 \cdot 4 = 2400$ bits at once. If we are using ASCII representation of English letters, we can encrypt 300 letters at once. The advantage of using more giant cubes is that we make both brute force attacks we discussed earlier more challenging to succeed. To begin with, when the side length of the Rubik's Cube gets greater, we permute much more bits. For C_{10}, we permute 2400 bits, and thus the attacker has to search through group S_{2400} which has 2400! elements. For an arbitrarily large cube C_n, we permute $24 \cdot n^2$ bits. It follows that if the attacker wants to search through the permutation group, he/she has to go through $(24 \cdot n^2)!$ possibilities. Therefore finding the inverse of the permutation will require much more computations as the side length of the cube increases. Besides, more giant cubes have more than just 18 fundamental moves. For example, C_{10} has 180 fundamental moves. Since each time there are much more fundamental moves to select, finding the inverse of the key will get a lot harder too.

There are also two vulnerabilities of this approach. The first one is that we still cannot encrypt an arbitrarily long message. Once we fix a cube length, say 20, we

do not want to change it during the communications. The reason is that, while modifying the size of the cube, we are changing the key space as well. That is the fundamental moves we can select for C_3 and C_{20} are obviously different. However, for private key encryptions, we commonly exchange the secret key once before building the secure channel between parties, and we use the same key for all of the following communications. Secondly, we may end up wasting a lot of space when we need to encrypt a relatively short message since the cube size is fixed.

A more general approach that works for message with arbitrary length l is called the cipher block chaining (CBC) mode, which the is the most commonly used mode of operation. Let us first introduce the operation "XOR" is denoted as $\oplus$ and it is also called "exclusive or". It gains the name "exclusive or" because it excludes the case when both operands are true. "XOR" is a logical operation that outputs true only

Input A	Input B	A $\oplus$ B
1	1	0
1	0	1
0	1	1
0	0	0

Table 2.1: XOR truth table

when inputs differ. We build the truth table for it as shown in Table 2.1.

In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted as shown in Figure 2.15. Hence each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, we can use an initialization vector to XOR with the first block. Assume in total there are k blocks of plaintexts $(m_0, m_1, ..., m_k)$ needed to be encrypted, the output will be k blocks of ciphertexts $(c_0, c_1, ..., c_k)$. Giving all the ciphertexts, to

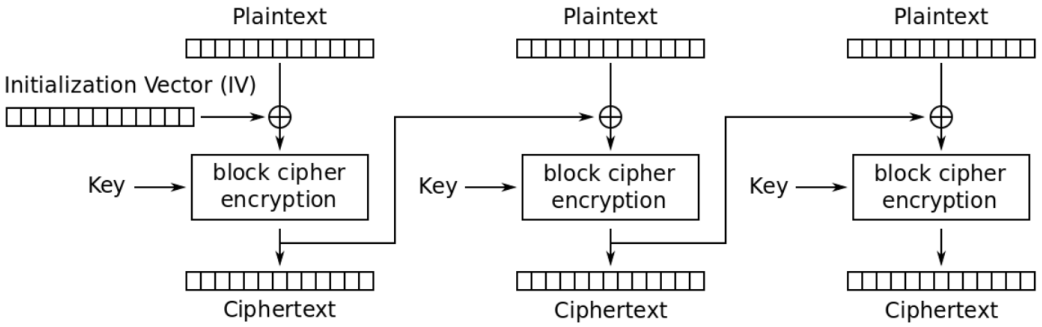

Figure 2.15: CBC mode encryption[7]

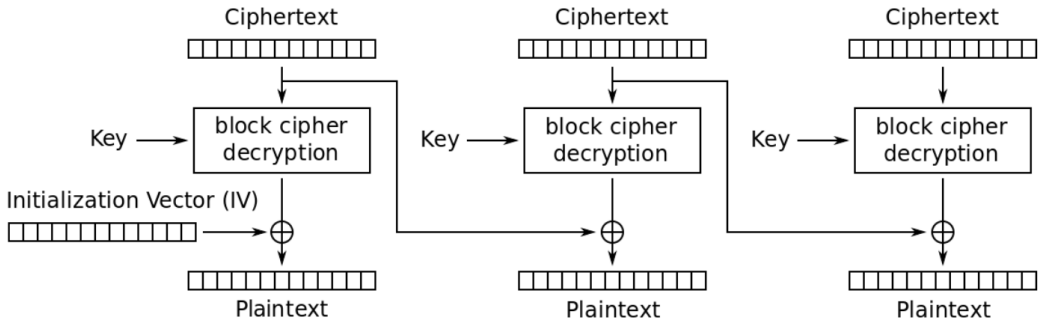

Figure 2.16: CBC mode decryption[6]

retrieve m_x where $2 \leq x \leq k$, we decrypt c_x with the key and then XOR the result with c_{x-1} to get m_x back as shown in Figure 2.16. The first ciphertext c_1 is unique since we need to XOR its decryption result with the initialization vector to recover m_1. Though this general approach is easy to manage, it has some potential vulnerabilities such as one bit wrong in the x_{th} ciphertext will generate errors in all blocks after it.

Finally, for both approaches discussed above, we have to deal with one more issue. In most times, we probably will not have a message that fits perfectly with the encryption scheme we have. Suppose we have a message m with length of 2000 bits and we are using C_{10} to encrypt it. Recall that C_{10} offers 2400 spaces for bits and thus we have 400 spare spaces. We want to agree on a simple padding algorithm so that we do not have to leave some spaces blank. To pad the message m, we simply add a

one at the end of it and add 399 more zeros to get the desired length. Notice that we have to complete the padding before encrypting the message. After decryption, the receiver simply removes all trailing zeros and removes one extra 1 to retrieve the original plaintext. This padding scheme will work for the CBC mode as well. Assume we are chaining a list of C_3 to encrypt m, we then need $\lceil \frac{2000}{216} \rceil = 10$ blocks, which offers $216 \cdot 10 = 2160$ spaces. Similar to above, we have to add a one and 159 more zeros at the end of m to get the desired length. Note that if our message splits up evenly into k blocks, we need to add in the $k + 1$th block for just the padding.

In this chapter, we have designed a symmetric encryption scheme that avoids some apparent shortcomings and can encrypt arbitrarily long messages. In the next chapter, we will analyze our protocol under a more formal framework and see that further improvements are needed.

Chapter 3

Security of an encryption protocol

In this chapter, we give the formal definition of security of symmetric encryption and test our developed Rubik's Cube encryption protocol against it. We make essential tweaks to the Rubik's Cube encryption to make it fit in the definition, and we generate some statistics to show that the final improved encryption produces a good shuffling.

3.1 What is security?

We denote the Rubik's Cube encryption protocol as $\Pi_{RC} = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$. Let us first review procedures of Π_{RC} by listing out details of each component, assuming that we are still using a normal 3 by 3 by 3 cube.

- **Gen** takes an integer n as input representing the number of fundamental moves in the key. It then draws one move from the 18 fundamental moves uniformly at random n times and denotes the series of moves as k. Afterwards, it picks one permutation from S_{216}, the permutation group on 216 elements, uniformly at random and denotes the permutation as p. Finally, **Gen** outputs (k, p).

- **Enc** takes a tuple (k, p, m), where m has length of 216 bits, as the input. Then it fills an empty cube with m in the following order: top face $\rightarrow$ front face $\rightarrow$ right face $\rightarrow$ down face $\rightarrow$ back face $\rightarrow$ left face. Recall Figure 2.10, for each face, **Enc** starts to fill at the left up corner cubie, and once it completes a row, it goes to the next row until it reaches the right down corner cubie. For each cubie, **Enc** fills it with the order: left up corner $\rightarrow$ right up corner $\rightarrow$ right down corner $\rightarrow$ left down corner. After filling the cube, **Enc** permutes the bits on the cube using p and applies the first movement in k to the cube. Then **Enc** permutes the bits again using p and applies the second movement in k to the cube. **Enc** repeats itself until it reaches the last move in k. Finally, **Enc** reads out the bits the same order as they were put in and outputs the encrypted bits c. We denote this process by $\mathbf{Enc}_{kp}(m)$.

- **Dec** takes a tuple (k, p, c) where c has length of 216 bits, as the input. It then fills an empty cube with c the same way as **Enc**. Then **Dec** finds inverses of k and p and denotes them as k' and p'. Afterward, **Dec** applies the first movement in k' to the cube and permutes the bits on the cube using p'. **Dec** repeats itself until it reaches the last move in k' and completes last permutation using p'. Finally, **Dec** reads out the bits the same order as they were put in and outputs the decrypted bits m. We denote this process by $\mathbf{Dec}_{kp}(c)$.

In the last chapter, we claimed our improvements would make the encryption scheme stronger against the brute force attacks. However, does that mean we can conclude that the Rubik's Cube encryption is secure? So far, we have not yet defined what we mean by the security of an encryption protocol. Ideally, a safe encryption protocol will leak zero bit of information of the plaintext.Though this sounds like a stringent

requirement, it is necessary for any secure encryption protocol to satisfy in real life applications. Suppose we are voting anonymously to select the class president between two candidates. If the encryption system we use leaks information about one letter in the plaintext and if that letter happens to exist in one candidate's name but not the other, people who monitor the Internet will then be able to tell our decisions. However, as one can imagine, deciding the amount of information leaked can be a difficult task.

We do so for an encryption protocol $\Pi = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$ by defining an interactive experiment E_Π between Alice, the adversary and Bob, the holder of Π. The

Alice	Bob
Generates messages m_0, m_1, both of length n that can be accepted by Π. Sends m_0, m_1 to Bob.	
$\xrightarrow{m_0, m_1}$	
	• Randomly selects m_0 or m_1. • Uses **Gen** to generate a key k. • Encrypts the selected message using k to form the *challenge ciphertext* c. • Sends c to Alice.
$\xleftarrow{c}$	
Guesses whether Bob encrypted m_0 or m_1 to form c.	

Table 3.1: Procedure of experiment E_Π

details of E_Π is shown in Table 3.1. If Alice guesses correctly on which message was encrypted, we say that Alice wins the experiment.

Notice that in above experiment, Alice has the freedom to pick any m_0 and m_1 as long as both of them have the desired length. Therefore, if Π is not secure and leaks information when given certain inputs, Alice can always choose those inputs and win this experiment with a pretty good chance. However, if Π is truly secure, in each run of the experiment, Alice has to randomly guess which message was encrypted. Therefore when Π is secure, Alice should have exactly $\frac{1}{2}$ probability of winning the experiment E_Π. We now can give the formal definition of security of private-key encryption schemes.

Definition 3.1. ***Security of private-key encryption scheme***
A private-key encryption scheme $\Pi = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$ *is secure, if for all adversaries, they have exactly* $\frac{1}{2}$ *chance to win the experiment* E_Π.

In real life, we only consider adversaries running in limited time, and therefore we allow the adversary to win with probability negligibly better than $\frac{1}{2}$. While not diving into details of negligible functions, we can solely consider it as a tiny error term.

3.2 Security of Rubik's Cube encryption

Our encryption system Π_{RC} is not secure under Definition 3.1. We can slightly modify Alice's strategy to show how she can always win the experiment $E_{\Pi_{RC}}$. The behavior of Alice is altered as shown in Table 3.2. The reason our improved encryption system quickly failed the definition of security is that it solely offers a way to permute the plaintexts without modifying any of the bits. We see that the challenger cipher c must be either an all zero string or an all one string because Π_{RC} only permutes the plaintext. Hence, by using the strategy described in Table 3.2, Alice can win the

Alice	Bob
Sends messages m_0, m_1 with $m_0 = \underbrace{000...0}_{216}$ and $m_1 = \underbrace{111...1}_{216}$ to Bob.	
$\xrightarrow{m_0, m_1}$	
	Same steps as in Table 3.1.
$\xleftarrow{c}$	
Guesses m_0 was encrypted if c is an all-zero string and guesses m_1 was encrypted if c is an all-one string.	

Table 3.2: Procedure of experiment $E_{\Pi_{RC}}$

experiment with a probability of 1. When interacting with the encryption protocol, Alice, can always find the message encrypted by merely counting the number of zeros and ones in the messages.

One way for us to make further modifications to provide more security under this definition is to bring in some randomness in the encryption procedure. Intuitively, instead of using all six faces of the Rubik's Cube to hold plaintext, we use only five faces for the actual plaintext and we use one face to hold some random bits. That means instead of encrypting 216 bits, we encrypt $36 \cdot 5 = 180$ bits at a time. Suppose we use the top face, front face, right face, down face and back face for the actual message. We then can use the left face for the random bits. However, solely doing this will not be enough. As shown in Table 3.3, we can again modify Alice's strategy and show how she can break this improved schema.

For an input with all ones, the output will have a minimum of 180 ones and the all-zero input will have a maximum of 36 ones. Therefore the adversary can still distinguish the two input messages by simply counting the number of ones and zeros.

Thus, to fully utilize the randomness, we want to spread it across the cube during

Alice	Bob
Sends messages m_0, m_1 with $m_0 = \underbrace{000...0}_{180}$ and $m_1 = \underbrace{111...1}_{180}$ to Bob.	
$\xrightarrow{m_0, m_1}$	
	Same steps as in Table 3.1.
$\xleftarrow{c}$	
Guesses m_0 was encrypted if c has more or equal to 180 zeros and guesses m_1 was encrypted if c has more or equal to 180 ones.	

Table 3.3: Procedure of experiment $E_{\Pi_{RC}}$

the encryption. We can achieve such task by performing XOR operation between the left face, which holds the random bits and the rest of the faces. To make the randomness spread evenly, we want to perform this XOR operation every time before we apply the permutation and the movement to the cube. In Table 3.4, we display the first five bits of each cube face and build a minimum example to illustrate this process. Notice that the bits on the left face remain unchanged. We can encrypt Einstein's

Cube face	Before XOR	After XOR
Up face	00000...	11111...
Front face	00001...	11110...
Right face	00011...	11100...
Down face	00111...	11000...
Back face	01111...	10000...
Left face	11111...	11111...

Table 3.4: Example of XOR operation

quote again with the revised Rubik's Cube encryption. We need to first cut the size of the input down to 180 bits and we need to generate 36 continuous random bits.

0100010001001111010011100100111101010100010101110100111101010010010100100101100101000001010000100100111101010101010101000101100101001111010101010100100010001001001010001100100000000010111001010010100010111100010

As displayed above, the leading bits are the actual input and the random bits are colored in blue. We again use the same key "$R\,U\,B2\,L'\,D2\,F\,R'\,D\,B\,L2$" to encrypt the cube. Each time before we apply one move to the cube, we first perform the XOR operation and then shift the bits. After we use all moves in the key, we get the result:

0110001111011111100001101110000100101010111010110010000111011110110001000110100111101101010001101111010010100111011000011100011000001010111110010011101011100100010100111000011000111101010111010101100110000010101000000

Along with the random bits, the plaintext has 122 zeros and 94 ones but after the encryption, the ciphertext has 107 zeros and 109 ones. In this example, we find the number of ones and zeros become pretty balanced. If the XOR operation has the same effect on balancing the bits for all input strings, the counting method Alice can use as strategy in the experiment Π_{RC} will become much less feasible.

3.3 Analysis of Rubik's Cube encryption

The implementation of the Rubik's Cube encryption system in python is attached in the Appendix. In order to illustrate some insights of how well this encryption can shuffle the input bits, we use it to generate some statistics. Also, by looking at those results, we are able to make some suggestions for picking keys with the proper length.

The first interesting thing we can keep track of is how bits travel when we apply a random key on the cube. Recall the order of putting the bits in by faces is: up face

→ front face → right face → down face → back face → left face. Each face holds 36 bits as displayed in Figure 2.10. Thus the bits on up face will be labeled from 1 to 36, the bits on front face will be labeled from 37 to 72 and so on. Therefore we use number 1 through 216 to represent locations of all bits.

In Figure 3.1, we display how two adjacent bits at location 1 and location 2 travel under 20 random fundamental moves. Notice that, in the experiment, we used the bit shift as the permutation. The blue line in the graph tracks the bit at location 1

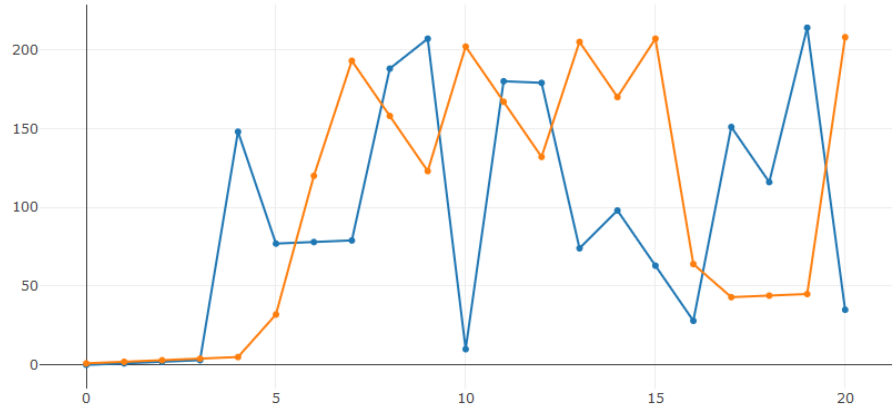

Figure 3.1: Bit locations

while the orange line in the graph tracks the bit at location 2. The x-axis in the graph represents the number of movements and the y-axis represents the location. These two bits end up at location 35 and location 208 individually. Though they start to be 1 space part, they end up being 173 spaces apart. We stimulated this process with 100 random series of 20 fundamental moves. Each time we recorded the final distance between the same two bits and denote the value as FD. Among the 100 distances FD we collected, 73 were unique and the average of FD is about 71 spaces. To determine the proper key length, we repeat the process by picking 100 random series for each key length and the results are displayed in Table 3.5. It seems that mean of FD, median

Key length	Mean FD	Median FD	Max FD	Min FD	Unique FD
8	56.2	43.5	198	1	61%
9	58.8	47	194	1	63%
10	60.4	51	200	1	66%
12	66.7	55.5	212	1	68%
15	68.7	60	214	1	70%
20	70.6	63	208	1	73%
30	73.3	65	207	1	76%

Table 3.5: Statistics on FD

of FD and number of unique FD has a positive relationship with the key length, but as key length gets longer, those values increase slower. Also we notice that even if we shuffle the cube more, two adjacent bits can still end up to be adjacent. But more fundamental moves do provide more possible final distances. As expected, we suggest using a key with longer length in general provides more security. We will discuss more in the next a couple of paragraphs to find an exact number as the suggested value.

In addition, we can check how the ciphertext differs from the plaintext. That is, we want to know if the XOR step does a good job mixing the zeros and ones. Let us revisit the vulnerability we found earlier. Suppose we have an all-zero string input and the bits on the left face are randomly generated. Each time we apply a random key formed by 10 fundamental moves to the cube and find the number of zeros we end up having in the ciphertext. Let us denote this value as NZ, short for number of zeros. We find the average of NZ is about 101 when we use 10 fundamental moves. We can increase this number to see how many moves we need to make about half of the bits to be ones in the ciphertext. From Table 3.6, we find that when we increase the key length to 25 fundamental moves, the average of NA increases to 108.3. In addition, there were actually equal number of zeros and ones or more ones than zeros in 52 times among the 100 runs. This result provides evidence that the XOR operation can

Key length	Mean NZ	Median NZ	Max NZ	Min NZ	$NZ \leq 108$
10	100.7	102	124	56	23
15	105.1	105.5	126	80	41
20	107.4	108	125	91	51
25	108.3	107	126	91	52
30	108.5	109	127	93	57

Table 3.6: Statistics on NZ

successfully hide the fact that an all zero string was encrypted. Results above suggest that a key with length of 25 fundamental moves may give us reasonable security and efficiency for the Rubik's Cube encryption.

Notice that if the random bits happen to be all zeros while we are inputting an all-zero string, the entire string will stay as all zeros no matter how we shuffle it. In the experiment, Alice can use this vulnerability to attack the improved encryption protocol. However, there are in total $2^{4 \cdot 3 \cdot 3} = 2^{36}$ set of bits the left face of C_3 can generate and only one set among those contains all zeros. If we use an arbitrary large cube C_n, the left face of C_n can generate 2^{4n^2} different set of bits. Hence, Alice has a really small chance to win even if she knows about this vulnerability. We claim that Alice has merely a negligible chance to win the experiment more than half times by using the bit counting strategy.

One other thing that is commonly measured to test encryption protocols is called *diffusion*, which means that if we flip a single bit of the plaintext and encrypt it again, then statistically, half of the bits in the ciphertext should change. We fix a random key with length of 25 fundamental moves and we create two messages that differ at exactly one bit. We encrypt both message and count how many bits between them are different. Let us denote this value as BD, shorten for bit difference. Similar to above, we repeat this process 100 times for each unique key lengths. From Table 3.7,

Key length	Mean BD	Median BD	Max BD	Min BD
25	70.6	87	120	1
30	79.4	95.5	123	1
40	93.2	104	122	1
50	98.2	108	125	1

Table 3.7: Statistics on BD

we see that to provide sufficient diffusion, we need the key length to be about 50.

In this chapter we have introduced the formal definition of security of encryption protocols. To make the Rubik's Cube encryption stronger under this definition, we introduced randomness in the encryption procedure. We also found evidence that key length of 30 may give reasonable security and efficiency for the encryption.

In next chapter, we will explain some known results about the size the group G_n and the explicit structure of G_3. This will lay the ground work for using Rubik's Cube as the basis for a key exchange system in Chapter 5.

Chapter 4

More group structure of the cube

In this chapter, we dive into more details about the group structure of Rubik's Cubes and explain how to find the group size of an arbitrarily large Rubik's Cube. While giving the definition of the semi-direct group, we illustrate the explicit structure of G_3 to build up for the next chapter.

4.1 Size of the Rubik's Cube group

In Chapter 2, we mentioned that there exist some illegal states of the Rubik's Cube C_3 and thus the group G_3 constructed by states of C_3 is a subgroup of S_{48}. We now want to show how to find the size of group G_3 and we expand the method to illustrate how to calculate size of group G_n constructed by an arbitrary large cube C_n.

Recall Figure 2.1, we showed that there are 8 corner cubes in C_3. The 8 corner cubes can freely exchange their locations and they can be arranged in 8! ways. When we fix locations of the 8 corner cubes, each of them can be arranged in 3 different orientations, giving 3^8 possibilities for each ordering of the corner pieces. Similarly,

there are 12 edge cubes that can freely exchange their locations and the can be arranged in 12! ways. When we fix locations of the 12 edge cubes, each of them can be arranged in 2 different orientations, giving 2^{12} possibilities for each permutation of the edge cubes. However, further reductions are required due to the structure of the Rubik's Cube.

The reason we need to further reduce the group size is explicitly explained in Chapter 11 in paper[2]. We will not repeat all the details presented there but give the essential part of the argument. For the 8 corner cubes of C_3, only 7 of them can be oriented independently, meaning the orientation of the 8th corner cube depends on the preceding seven. Hence, when we fix locations for all 8 corner cubes, they can have at most 3^7 possible different orientations. Similarly, when we determine orientations for 11 of the 12 edge cubes, the orientation of the last edge cube will be fixed. It follows that there are 2^{11} possible orientations for each permutation of the edge cubes. So for now, the order of C_3 can be calculated as $8! \cdot 3^7 \cdot 12! \cdot 2^{11}$. In [2], a labelling system on the cube was created, and the authors proved that any permutation of the edges must be an even permutation. Therefore we must reduce this number by half. Finally, we find the order of C_3 is $8! \cdot 3^7 \cdot 12! \cdot 2^{10} = 43,250,003,274,489,856,000$.

For a hint of the proof for above statements, recall that in Chapter 2, we defined each valid state as a shuffling result of a series of fundamental moves. By analyzing properties of those permutations, we can obtain a good understanding on how to determine if a state of C_3 is valid.

Now we can explain how to find group order of G_n for a arbitrarily large cube C_n. The calculation for corner cubes will stay true for any cube C_n but the calculation for edge cubes will change since cubes with different side lengths have different number of edge cubes and for larger cubes, there will be two types of edge cubes. We also need

to introduce a couple more different small cubes with single visible colored piece. We can use C_7 to illustrate all types of small cubes. Consider Figure 4.1 as one face of the cube. The small cubes colored in blue are the corner cubes and those colored in yellow are the fixed center cubes. Those small cubes behave the same as they are in C_3. In

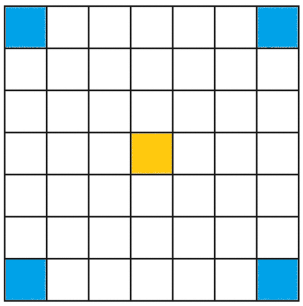

Figure 4.1: Center and corner cubes

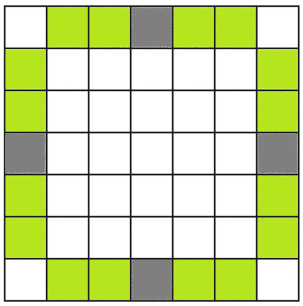

Figure 4.2: Edge cubes

Figure 4.2, we are showing the two types of edge cubes. The ones colored in green are called *wing edges* and the ones colored in grey are called *middle edges*. Clearly, when we shuffle the cube, the middle edge will only permute with other middle edges and the wing edges will permute with other wing edges. The middles edges in C_7 generate exactly the same amount of permutations and orientations as they are in C_3. We can further split the wing edges into two sets as shown in Figure 4.3. The yellow wing edges will only permute with other yellow wing edges and the blue ones will permute with other blue ones. For an arbitrary large cube C_n, we can calculate the number of sets of wing edges it has by finding $\lfloor \frac{n-2}{2} \rfloor$ and each set of wing edges has $2 \cdot 12 = 24$ small cubes. Due to the inner structure of Rubik's Cubes, different from the middles edges, each one of the wing edges has only one possible orientation when it is at fixed locations.

Besides the fixed centers, there are three more types of small cubes with only one visible piece we want to introduce. The ones displayed in Figure 4.4 are called the "+

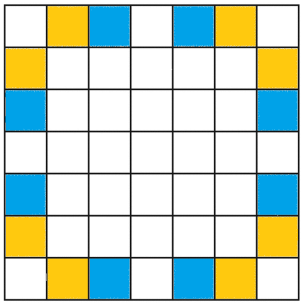

Figure 4.3: Wing edges

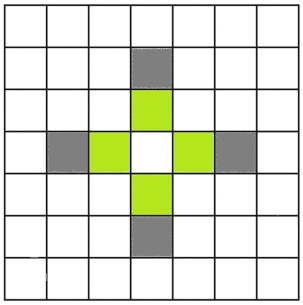

Figure 4.4: + centers

centers" since they form the + shape. Similar to the wing edges, they can be further split into two sets and those pieces only permute with same colored ones. Secondly, we want to show the "× *centers*". As displayed in Figure 4.5, those pieces form the × shape and they can be split to two sets as well. The number of sets of + centers and

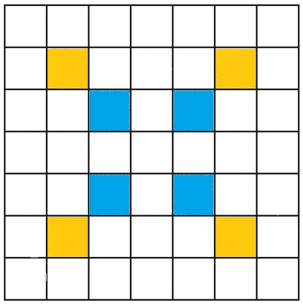

Figure 4.5: × centers

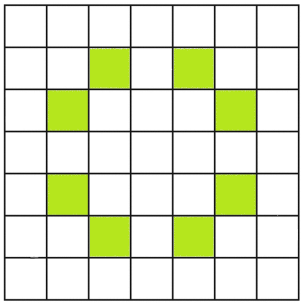

Figure 4.6: Oblique centers

× centers of an arbitrarily large cube C_n can be calculated by the equation defined above for wing edges, and each set of + centers or × centers contains $4 \cdot 6 = 24$ small cubes. Finally, the pieces that are not covered by the + shape and the × shape are denoted the oblique centers. They are displayed in Figure 4.6. Though these three type of centers pieces are different, they permute in a similar fashion; they can change their locations freely but not their orientations.

We can further differentiate odd cubes and even cubes. Since C_7 is an odd cube,

all types of small cubes exist within it. For an even cube, there will be no middle edges and "+ centers" since the fixed center piece does not exist. Therefore, there is no preferred orientation of an even cube, and some sequences of fundamental movements will be equivalent to rotating the entire cube in a three-dimensional space. Thus, the number of permutations is reduced by a factor of 24. This is because all 24 possible positions and orientations of the first corner are equivalent because of the lack of fixed centres. After addressing all types of small cubes and the difference between odd and even cubes, we now have all the tools we need to calculate group size of G_n for an arbitrary cube C_n. In Table 4.1, we show details of this calculation. Finally we can

Type of cube	Number of orderings
Corner cubes	All cubes can arrange their corners in $8! \cdot 3^7$ different ways.
Middle edges	All odd cubes can arrange their middles edges in $12! \cdot 2^{10}$ ways. Since even cube does not have middle edges, an arbitrary cube C_n has $(12! \cdot 2^{10})^{n \mod 2}$ ways to order its middles edges.
Wing edges	An arbitrary cube C_n has $(24!)^{\lfloor \frac{n-2}{2} \rfloor}$ ways to order its wing edges.
Fixed centers	Only odd cubes have fixed centers and since they are fixed, they do not any anymore possible orderings to the calculation.
"+ centers" "× centers" Oblique centers	Since those center pieces permute in a similar fashion, we can put them together to calculate the number of orderings they generate. The details of this calculation can be found in paper[4] and the result is given as $(\frac{24!}{(4!)^6})^{\lfloor (\frac{n-2}{2})^2 \rfloor}$.

Table 4.1: Number of permutation different type of small cubes generates

put everything together and the order of group G_n for states of an arbitrary Rubik's Cube C_n can be calculated by the following function:

$$f(n) = 8! \cdot 3^7 \cdot (12! \cdot 2^{10})^{n \mod 2} \cdot (24!)^{\lfloor \frac{n-2}{2} \rfloor} \cdot (\frac{24!}{(4!)^6})^{\lfloor (\frac{n-2}{2})^2 \rfloor} \cdot 24^{-((n+1) \mod 2)}$$

In Table 4.2, we show some calculations we obtained by using the equation above to

demonstrate how fast the corresponding group size increases when the side length of a Rubik's Cube increase. This quantifies our claim in Chapter 2 that the key space

Cube group	Group size
G_3	4.325×10^{19}
G_4	7.401×10^{45}
G_5	2.829×10^{74}
G_6	1.572×10^{116}
G_7	1.950×10^{160}

Table 4.2: Group size of different Rubik's Cubes

explodes as the size of our cube increases. The function $f(n)$ gives us the size of an arbitrary G_n, but the exact algebraic structure is not known for $n > 3$.

4.2 Semi-direct product of a group

The group structure of G_3 is well understood and it can be expressed as a semi-direct product of other common groups such as the cyclic group. Before we show how to represent C_3, we need to review some group theory knowledge, and some definitions here will help in understanding the key exchange protocol in Chapter 5 as well.

Definition 4.1. *A function ϕ from a group G to a group H is a **homomorphism** if ϕ preserves the group operation; that is if $\phi(ab) = \phi(a)\phi(b)$ for all $a, b \in G$.*

Definition 4.2. *An **isomorphism** is a homomorphism ϕ from a group G to a group H that is one-to-one and onto.*

Definition 4.3. *An **automorphism** is an isomorphism from a group G onto itself. The set of automorphisms of a group G is denoted by $Aut(G)$.*

Definition 4.4. *The **alternating group**, denoted A_n is the group of all even permutations in S_n.*

Definition 4.5. *Let G be a group and H be a subgroup of G, the subgroup H is a **normal subgroup** of G, denoted by $H \triangleleft G$, if $\forall g \in G$, $gH = Hg$.*

Definition 4.6. *Let G and H be two groups. Then the **direct product** of G and H is the group $G \times H$ under the operation $(g_1, h_1) \cdot (g_2, h_2) = (g_1 g_2, h_1 h_2)$ where $g_1, g_2 \in G$ and $h_1, h_2 \in H$.*

Definition 4.7. *Let H_1 and H_2 be two subgroups of G, then $G = H_1 \rtimes H_2$ is a **semi-direct product** if:*

1. *$G = H_1 H_2$.*
2. *$H_1 \cap H_2 = e$, where e is the identity element of G.*
3. *$H_1 \triangleleft G$.*

We can now describe the group structure of G_3. A complete proof can be found in [2], but we will sketch the details here. Let us consider two subgroups of G_3. First the subgroup H_1 of cube orientations, which leave the position of every small cube fixed but can change their orientations. One can show this subgroup is a normal subgroup of G. It can be represented by moves that flip a few edges or twist corners. For example, the move "$R\,U\,D\,B2\,U2\,B'\,U\,B\,U\,B2\,D'\,R'\,U'$ " flips two edges. The structure of this group is $\mathbb{Z}_3^7 \times \mathbb{Z}_2^{11}$ since the group of rotations of each corner is a cyclic group of order 3 and we mentioned that we can determine orientations of 7 among 8 corner cubes freely and the last one will be fixed. The argument for flipping edges is exactly the same.

In addition, we take the subgroup H_2 which permutes the positions of the small cubes but does not change their orientations. Similar to above, this group can be represented by a direct product of two subgroups, which are the group of permutations on the corners S_8 and the group of even permutations on the edges A_{12} (because of the even parity). Hence the structure of H_2 is $S_8 \times A_{12}$. Notice that here we do not mind if H_2 is a normal subgroup or not.

Clearly $H_1 \cap H_2 = e$, where e is the identity in G_3, since H_1 keeps all small cubes fixed but changes their orientations, whereas H_2 always moves the cubes. Since H_1 and H_2 account for all possible change on the cube C_3, H_1H_2 can form the entire group G_3. It follows that the cube group G_3 is isomorphic to the semi-direct product of these two groups:

$$G_3 = H_1 \rtimes H_2 \cong (\mathbb{Z}_3^7 \times \mathbb{Z}_2^{11}) \rtimes (S_8 \times A_{12})$$

By finding the group order of above semi-direct product, we will again retrieve the same result we found earlier in this chapter. For instance, the largest order of an element in G_3 is 1260 and one such element is "$R\,U2\,D'\,B\,D'$ "[1]. However, we have not yet found a efficient method to generalize those results to other Rubik's Cube groups G_n.

In this chapter, we introduced methods to find group size of G_n of an arbitrary large cube C_n. The explosion of the group sizes we observed suggests that larger cubes may make brute force attacks more computationally expensive. Although not known for larger cubes, we are able to give the isomorphism class for G_3. Since this additional structure is known, it could open up the door to opportunities to further reduce computation cost of the brute attacks on encryption based on C_3.

Chapter 5

Key exchange protocol

The private key system we developed in Chapter 2 and 3 required the two parties to possess the same shared key. One of the major development in modern cryptography is the public key system that allows parties to securely exchange a shared key to use in a symmetric system. In this chapter, we build a Rubik's Cube key exchange protocol using a structure similar to Diffie-Hellman key exchange.

5.1 Diffie-Hellman key exchange

As we mentioned before, communication between two parties using private-key encryption requires that they first exchange keys by some secure channel. Diffie-Hellman key exchange, shorten as DHKE, is a method of securely exchanging cryptographic keys over a public channel, and our Rubik's Cube key exchange protocol builds upon it.

1. Alice and Bob agree on a arbitrary finite Abelian group G of order n and an element $g \in G$. Both group G and the element g will be made public to everyone. (The group G is written multiplicatively.)

2. Alice picks a random natural number a, where $1 \leq a < n$, and sends $A = g^a$ to Bob. (a remains secret to everyone besides Alice.)

3. Bob picks a random natural number b, where $1 \leq b < n$, and sends $B = g^b$ to Alice. (b remains secret to everyone besides Bob.)

4. Alice computes $S_a = A^b$ and Bob computes $S_b = B^a$.

We claim $S = S_a = S_b$ because $S_a = A^b = (g^a)^b = g^{ab} = (g^b)^a = B^a = S_b$. This ensures that Alice and Bob end up with the same shared value. In Table 5.1, we display each party's knowledge on various values assuming the existence of an eavesdropper. We see that the security of the DHKE protocol is based on that given g, g^a, g^b, finding

Value	Alice	Bob	Eavesdropper
G	✓	✓	✓
g	✓	✓	✓
a	✓	✗	✗
b	✗	✓	✗
A	✓	✓	✓
B	✓	✓	✓
S	✓	✓	✗

Table 5.1: Diffie-Hellman secrecy table

a or b is difficult and thus finding the shared value g^{ab} is hard as well. This problem is known as the *discrete logarithm problem*. Though discrete logarithms are quickly computable in a few special cases, no efficient method is known for computing them in general.

We want to build a minimal example with the multiplicative group to walk through the details of DHKE. Suppose Alice and Bob agree to use a modulus $p = 11$ ($G = \mathbb{U}_{11}$), and a base $g = 2$ (which generates G).

1. Alice picks a random natural number $a = 3$, and sends $A = g^a$ to Bob.
   $A = 2^3 \mod 11 = 8 \mod 11 = 8$

2. Bob picks a random natural number $b = 7$, and sends $B = g^b$ to Alice.
   $B = 2^7 \mod 11 = 128 \mod 11 = 7$

3. Alice computes $S = B^a = 7^3 \mod 11 = 343 \mod 11 = 2$

4. Bob computes $S = A^b = 8^7 \mod 11 = 2097152 \mod 11 = 2$

5. Alice and Bob now share a secret, the number 2.

Of course, much larger values of a, b, and p would be needed to make the DHKE secure. In real life applications, p should have a length of 1024 bits or even longer and g should have large prime order. The most common used group by the DHKE protocol is the multiplicative group of integers modulo p, with the elliptic curves being a significant variant.

5.2 Rubik's Cube key exchange

Notably, the original implementation of the DHKE requires the selected group G to be Abelian but the Rubik's Cube group is not since the fundamental moves do not all commute with each other. By extracting the ideas presented in the paper[5], we can design a DHKE-like protocol that helps us securely exchange information between two parties but does not require the group G to be Abelian.

Assume that we have two parties, Alice and Bob, who want to share the secret key. Let them agree on using the Rubik's cube group G_3. An element (one state)

$g \in G_3$ is chosen and made public as well as an arbitrary automorphism ϕ of G_3. Alice chooses a private number $a \in \mathbb{N}$ and Bob chooses a private number $b \in \mathbb{N}$.

1. Alice picks a natural number $a > 1$ and computes $A = \phi^{a-1}(g) \cdots \phi^2(g) \cdot \phi(g) \cdot g$ and sends this value to Bob.

2. Bob picks a natural number $b > 1$ and computes $B = \phi^{b-1}(g) \cdots \phi^2(g) \cdot \phi(g) \cdot g$ and sends this value to Alice.

3. Alice computes her key $K_A = \phi^a(B) \cdot A$.

4. Bob computes his key $K_B = \phi^b(A) \cdot B$.

We claim that Alice and Bob share the same secret key. Alice's shared key is:

$$K_A = \phi^a(B) \cdot A = \phi^a(\phi^{b-1}(g) \cdots \phi(g) \cdot g) \cdot (\phi^{a-1}(g) \cdots \phi(g) \cdot g) \tag{5.1}$$

$$= (\phi^a(\phi^{b-1}(g)) \cdots \phi^a(\phi(g)) \cdot \phi^a(g)) \cdot (\phi^{a-1}(g) \cdots \phi(g) \cdot g) \tag{5.2}$$

$$= (\phi^{a+b-1}(g) \cdots \phi^{a+1}(g) \cdot \phi^a(g)) \cdot (\phi^{a-1}(g) \cdots \phi(g) \cdot g) \tag{5.3}$$

$$= \phi^{a+b-1}(g) \cdots \phi(g) \cdot g \tag{5.4}$$

We could go from step (5.1) to step (5.2) because of the operation preserving feature of automorphism, which is $\phi(g_1 g_2) = \phi(g_1)\phi(g_2)$. The shared key Bob holds can be calculated in a similar fashion:

$$K_B = \phi^b(A) \cdot B = \phi^b(\phi^{a-1}(g) \cdots \phi(g) \cdot g) \cdot (\phi^{b-1}(g) \cdots \phi(g) \cdot g)$$

$$= (\phi^{b+a-1}(g) \cdots \phi^{b+1}(g) \cdot \phi^b(g)) \cdot (\phi^{b-1}(g) \cdots \phi(g) \cdot g)$$

$$= \phi^{b+a-1}(g) \cdots \phi(g) \cdot g = K_A$$

Notice that the Diffie-Hellman secrecy table displayed in Table 5.1 still holds true for above settings. If we pick ϕ of G_3 as $\phi(g) = g$, we can discover that, different from the standard DHKE, correctness of the Rubik's Cube key exchange is based on the equality $g^a \cdot g^b = g^b \cdot g^a = g^{a+b}$.

Let us show an example of how to use Rubik's Cube key exchange protocol. The group we are using is G_3 and this information is known by public. We also pick a public automorphism ϕ to be a right conjugation by R. That is $\phi(g) = RgR^{-1}$. Finally we pick a public group element $g = U$ (the state we reach to by applying move U to a solved cube.)

1. Alice picks a random natural number $a = 4$, and sends A to Bob.

$$\begin{aligned} A &= \phi^3(g) \cdot \phi^2(g) \cdot \phi(g) \cdot g \\ &= R^3(g)R^{-3}R^2(g)R^{-2}R(g)R^{-1}g \\ &= R^3(g)(R^{-1}g)(R^{-1}g)(R^{-1}g) \\ &= R^3(g)(R^{-1}g)^3 \end{aligned}$$

2. Bob picks a random natural number $b = 8$, and sends B to Alice.

$$\begin{aligned} B &= \phi^7(g) \cdot \phi^6(g) \cdots \phi(g) \cdot g \\ &= R^7(g)R^{-7}R^6(g)R^{-6} \cdots R(g)R^{-1}g \\ &= R^7(g)(R^{-1}g)(R^{-1}g) \cdots (R^{-1}g) \\ &= R^7(g)(R^{-1}g)^7 \end{aligned}$$

3. Alice computes $K_A = \phi^4(B) \cdot A$ and finds the following result.

$$\begin{aligned}\phi^4(B) \cdot A &= \phi^4(R^7(g)(R^{-1}g)^7)\ R^3(g)\ (R^{-1}g)^3 \\ &= R^4R^7(g)(R^{-1}g)^7R^{-4}R^3(g)(R^{-1}g)^3 \\ &= R^{11}(g)(R^{-1}g)^7(R^{-1}g)(R^{-1}g)^3 \\ &= R^{11}(g)(R^{-1}g)^{11}\end{aligned}$$

4. Bob computes $K_B = \phi^8(A) \cdot B$ and finds the following result.

$$\begin{aligned}\phi^8(A) \cdot B &= \phi^8(R^3(g)(R^{-1}g)^3)\ R^7(g)\ (R^{-1}g)^7 \\ &= R^8R^3(g)(R^{-1}g)^3R^{-8}R^7(g)(R^{-1}g)^7 \\ &= R^{11}(g)(R^{-1}g)^3(R^{-1}g)(R^{-1}g)^7 \\ &= R^{11}(g)(R^{-1}g)^{11}\end{aligned}$$

5. Alice and Bob now share a secret series of fundamental moves of C_3.

$$K_A = K_B = R^{11}(U)(R^{-1}U)^{11}$$

Since R^{-1} is the move that reverses R, we can simply rewrite it as R'. Finally the move Alice and Bob shares is $\underbrace{R...R}_{11}U\ \underbrace{R'U...R'U}_{11}$. To get more complex result in real applications, we could explore the group structure of $G_3 \cong (\mathbb{Z}_3^7 \times \mathbb{Z}_2^{11}) \rtimes (S_8 \times A_{12})$ to select more complicated g and ϕ at the beginning.

Most modern key exchange protocols help communicating parties to establish a commonly shared number that is used for the key with a symmetric system like AES.

We can add one more step in the key exchange protocol so that Alice and Bob can use their secretly shared moves to retrieve a number. Alice and Bob need to agree on a particular setting of C_3. We will refer this cube as *number exchange cube*. They both fill cubie on up face, down face, right face by ones and fill cubies on front face, back face, left face by zeros as shown in Table 5.2. After they retrieve the shared fundamental moves $K_A = K_B$, they apply that series of moves to the number exchange cube and then read out the bits as the binary number. Notice that here each cubie only holds one bit and we do not want the center cubies to hold any information since they are fixed. We also do not want to use all bits from the cube because then the enemy knows that we are generating a binary of length 48 where half bits are ones and half bits are zeros. We instead only take bits from three faces, the up face, the front face and the right face. So for an arbitrary large cube C_n we can use it to share a binary value with length of $3 \cdot (n^2 - (n \mod 2))$ bits.

To illustrate a detailed example of above procedure, let us first take a look at Table 5.2, which describes the starting position of the number exchange cube C_3 that Alice and Bob uses to retrieve the shared number. Both Alice and Bob apply the

Up face	11111111
Front face	00000000
Right face	11111111
Down face	11111111
Back face	00000000
Left face	00000000

Table 5.2: Starting position of the number exchange cube C_3

shared moves $\underbrace{R...R}_{11} U \underbrace{R'U...R'U}_{11}$ to the number exchange cube, and the bits on each face is displayed in Table 5.3. Remember that we only use bits from three faces, and those bits we want are colored in blue. Both Alice and Bob read them out in

Up face	00011010
Front face	00100001
Right face	11110001
Down face	11110100
Back face	01100000
Left face	11111110

Table 5.3: Final position of the number exchange cube C_3

order of: up face $\rightarrow$ front face $\rightarrow$ right face to retrieve the shared binary number $000110100010000111110001_2$.

In practice, symmetric keys are usually much longer. For example, the software FireVault on Mac OS uses AES with a 256-bit key to encrypt user's hard drive. We could use C_{10} to exchange the key since C_{10} are capable of holding $3 \cdot 10^2 = 300$ bits.

Even though we do not directly use numbers as the key to our Rubik's Cube encryption Π_{RC}, given a binary number we can still convert it to a series of fundamental moves. Suppose we have a set S which contains all base 18 numbers with only one digit. Thus $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, g, h\}$. In Table 5.4 find a one-to-one mapping between elements of S and the 18 fundamental moves. Alice and Bob can

0 - U	1 - F	2 - R	3 - D	4 - B	5 - L
6 - $U2$	7 - $F2$	8 - $R2$	9 - $D2$	a - $B2$	b - $L2$
c - U'	d - F'	e - R'	f - D'	g - B'	h - L'

Table 5.4: Mapping between set S and fundamental moves

convert the shared binary number to a base 18 number, $g5bff_{18}$. Then their commonly shared key will be "B' L $R2$ D' D' ". Notice that the biggest possible binary we can share using C_3 is $2^{24} - 1$. By converting that number to base 18, we get number $8fed99$. Since the maximum value on the first digits is 8, if we always use all 6 digits, the first movement will always be restricted. We thus simply ignore the first digit and

use last 5 digits to represent fundamental moves when the number happen to have 6 digits. When the number has less than 5 digits, we treat leading digits as zeros. In Table 5.5, we show the length of the key different C_n can share.

$C_3 - 5$	$C_4 - 11$	$C_5 - 17$	$C_6 - 25$	$C_7 - 34$
$C_8 - 46$	$C_9 - 57$	$C_{10} - 71$	$C_{15} - 161$	$C_{19} - 258$

Table 5.5: Maximum length of the Π_{RC} key C_n can share

In Chapter 3, we suggested to use a key with length 30. Hence choosing C_7 here might be most appropriate option. We now have completed the setting of the key-exchange encryption protocol and shown how to apply this to the actual Rubik's Cube encryption Π_{RC}.

5.3 Complete example

Let us put everything together and demonstrate a minimal example on how Alice and Bob can securely communicate with each other by using the Rubik's Cube encryption and the Rubik's Cube key exchange protocol on C_3. Suppose Alice now wants to send out "Hello Bob!"

Before the secure channel is built, Alice and Bob first exchange the shared key. They pick the public element $g = U\,R\,B\,D\,L'$ and let the public automorphism $\phi \in Aut(G_3)$ to be an inner-automorphism $\phi(g) = U2\,L2\,D'\,g\,D\,L2\,U2$, and agree to use the bit shift as the permutation p during encryption.

- Alice picks a random natural number $a = 23$, and sends A to Bob.

$$A = \phi^{22}(g) \cdots \phi(g) \cdot g$$

$$= (U2\,L2\,D')^{22}U\,R\,B\,D\,L'(D\,L2\,U2\,U\,R\,B\,D\,L')^{22}$$

- Bob picks a random natural number $b = 37$, and sends B to Alice.

$$\begin{aligned} B &= \phi^{36}(g)\cdots\phi(g)\cdot g \\ &= (U2\,L2\,D')^{36}U\,R\,B\,D\,L'(D\,L2\,U2\,U\,R\,B\,D\,L')^{36} \end{aligned}$$

- Alice computes $K_A = \phi^{23}(B)\cdot A$ and finds the following result.

$$\phi^{4}(B)\cdot A = (U2\,L2\,D')^{59}U\,R\,B\,D\,L'(D\,L2\,U2\,U\,R\,B\,D\,L')^{59}$$

- Bob computes $K_B = \phi^{37}(A)\cdot B$ and finds the following result.

$$\phi^{8}(A)\cdot B = (U2\,L2\,D')^{59}U\,R\,B\,D\,L'(D\,L2\,U2\,U\,R\,B\,D\,L')^{59}$$

- Alice and Bob now share a secret series of fundamental moves of C_3.

$$K = K_A = K_B = (U2\,L2\,D')^{59}U\,R\,B\,D\,L'(D\,L2\,U2\,U\,R\,B\,D\,L')^{59}$$

Both Alice and Bob apply the shared key K to the number exchange cube C_3. The initial bits and final bits on faces of C_3 are displayed in Table 5.6 and Table 5.7 separately. The binary number Alice and Bob both retrieve is $111011010100001010011011_2$. Then, they convert this binary value to a base 18 number, $8422hh_{18}$. Alice and Bob will omit the first digit since they agree to use only the last 5 digits. By looking at the mapping described in Table 5.4, they find the shared key is "$B\,R\,R\,L'\,L'$." Now

Up face	11111111
Front face	00000000
Right face	11111111
Down face	11111111
Back face	00000000
Left face	00000000

Table 5.6: Starting position of the number exchange cube C_3

Up face	11101101
Front face	01000010
Right face	10011011
Down face	00110000
Back face	00010110
Left face	10110111

Table 5.7: Final position of the number exchange cube C_3

Alice can convert her message to its binary representation.

$$01001000011001010110110001101100011011111$$

$$00100000010000100110111011000100010000 1$$

Alice no longer has to remove punctuations or capitalize all letters because the message will get changed during the encryption. Since this message contains merely 80 bits, and C_3 needs an input of 180 bits to encrypt. Alice needs to pad it first. The padded string is shown as below:

$$010010000110010101101100011011000110111100100000100001001100$$

$$111101100010001000011000000000000000000000000000000000000000$$

$$000000000000000000000000000000000000000000000000000000000000$$

After inputting the string, 36 random bits will be generated by the encryption protocol "111100001101010110101000010000011110," and those bits will be added to the end of the padded string. During the encryption process, before applying each fundamental move in the key, the encryption protocol first does a XOR between faces, and then shifts bits to the right by one space. After all moves are executed, the ciphertext is:

000111011001000100101100011101101110001010000000000100100100101010100001

010001000111011010000000100000010001101111111011000010000100110010011101

010101110000101000100101010000100110001110000011010001101011000110101010

Alice can convert this binary value back to its ASCII representation, which looks like " É,vΓÇ JíDvÇü √ L¥W %BcâF|¬ ," and sends it to Bob.

To decrypt Alice's message, Bob has to first find the inverse of the key "$B\,R\,R\,L'\,L'$", which will be "$L\,L\,R'\,R'\,B'$." Then he converts the received message back to its binary representation to retrieve the 216 bits and fills an empty C_3 with the bits. For each fundamental move in "$B\,R\,R\,L'\,L'$", Bob first applies the move to the cube, then shift bits back to the left by one space and finally performs the XOR operation between the left face and the rest of faces.

After all moves in the inverse of the key are applied to the cube, Bob will get the padded string. He removes all tailing zeros and one extra 1 at the end to get the plaintext in its binary representation.

010010000110010101101100011010001101111

001000000100001001101111011000100010000 1

By looking up the ASCII table, he gets Alice's message "Hello Bob!" To reply Alice's message, he can follow the same encryption process used by Alice, and he will keep using the shared key without exchanging new ones.

The example above illustrates how everything we developed ties together and serves as a complete encryption system that helps two parties to communicate with some protection.

Chapter 6

Conclusion

In conclusion, we successfully built a symmetric encryption protocol and a key exchange protocol using the Rubik's Cube. The steps we took demonstrated the general strategy of constructing encryption schemes: we first need to find a problem that is computationally hard, and we must analyze the problem and understand it well. We have shown the central role that a formal definition of security plays in the cryptography world. You must know what you are trying to achieve to deduce if you succeed or not. While we are not recommending commercial usage of this encryption system, we believe that it does provide a secure environment for two parties to communicate without leaking any visible information, given that the attacker possesses limited computational power such as one laptop.

Future work on this topic includes testing the Rubik's Cube encryption under even stronger definitions. Recall that in the security game described in Chapter 3, Alice can interact with the protocol only once before she guesses the encrypted message. But in the real world, once parties construct a secure channel, they will likely communicate multiple times. Thus under a stronger definition of security, we allow Alice to make

multiple requests and observe the results to catch potentially leaked information. Then if she notices any vulnerabilities in the trial queries, she can generate two corresponding messages to crack the protocol. For the key exchange protocol, we can expand this to an $n-$party key exchange protocol, rather than the two-party protocol we designed.

Bibliography

[1] David Joyner. "Adventures in group theory: Rubik's Cube, Merlin's machine, and Other Mathematical Toys". In: (2002).

[2] Janet Chen. "Group Theory and the Rubik's Cube". In: (2004).

[3] cryptography. *Merriam-Webster.com. Web. 15 Dec 2018.* Merriam-Webster, 2011.

[4] Christopher Mowla. "The Rubik's Cube". In: (2011).

[5] Maggie Habeeb et al. "Public key exchange using semidirect product of (semi)groups". In: *Lecture Notes in Computer Science*. Vol. 7954. Berlin, Heidelberg: Springer, 2013.

[6] White Timberwolf. *Decryption using the Cipher Block Chaining (CBC) mode*. June 2013. URL: `en.wikipedia.org`.

[7] White Timberwolf. *Encryption using the Cipher Block Chaining (CBC) mode*. June 2013. URL: `en.wikipedia.org`.

[8] *God's Number is 20*. URL: `www.cube20.org`.

Appendix

The implementation of the Rubik's Cube encryption is hosted on GitHub at https://github.com/Weiqi97/Cube-Crypto and the file structure is listed here:

```
cube_encryption
├── analyzers
│   ├── bit_analyzer.py
│   ├── key_analyzer.py
│   └── location_analyzer.py
├── encrypt_bit
│   ├── cube.py
│   ├── cubie.py
│   ├── encryption.py
│   └── face.py
├── encrypt_item
│   ├── cube.py
│   ├── encryption.py
│   └── face.py
└── helper
    ├── constant.py
    └── utility.py
```

Files in analyzers were used to generate statistics in Chapter 3. The folder encrypt_bit contains the Rubik's Cube encryption Π_{RC} we described in this thesis. It allows users freely pick the size of the desired Rubik's Cube while it takes in an English sentence as the input. This implementation uses CBC mode when the cube used cannot fit the entire plaintext.

The folder encrypt_item contains an implementation that is similar to the encryption Π_{RC} but does not require the input to be a string. As long as the input is a list of items, the encryption protocol will perform a shuffling on the elements of that list. Finally under helper folder, there are functions to convert strings to binaries and vice versa, as well as function that stimulates **Gen** and generates random keys with desired length. On GitHub, there is also a Jupyter notebook which contains sample runs of the encryption system.