A McEliece Cryptosystem, Using Permutation Error-Correcting Codes

Fiona Smith
College of Saint Benedict/Saint John's University

Follow this and additional works at: https://digitalcommons.csbsju.edu/ur_thesis

Part of the Mathematics Commons

Recommended Citation
https://digitalcommons.csbsju.edu/ur_thesis/35

This Thesis is brought to you for free and open access by DigitalCommons@CSB/SJU. It has been accepted for inclusion in CSB/SJU Distinguished Thesis by an authorized administrator of DigitalCommons@CSB/SJU. For more information, please contact digitalcommons@csbsju.edu.
A McEliece cryptosystem, using permutation error-correcting codes

A Distinguished Thesis

College of Saint Benedict and Saint John’s University

Department of Mathematics

by

Fiona Smith

April, 2024
Contents

1 Introduction 2

2 Preliminaries 4
  2.1 Permutation Groups ........................................... 4

3 Algebraic Coding Theory 10

4 The Problem 14

5 Srinivasan and Mahalanobis’s Cryptosystem 16

6 Our Cryptosystem 19

7 Programming the Process 22

8 Appendix: Developing Python Code 24

9 References 28
Acknowledgements

I would like to acknowledge the math department at the College of Saint Benedict and Saint John’s University for the support in my completion of this thesis. Particularly, I want to thank my thesis advisor Dr. Bret Benesh for his time and mentorship throughout this project, as well as Dr. Anne Sinko and Dr. Robert Campbell III, my other thesis readers who have provided helpful feedback in the revision process.
Chapter 1

Introduction

Cryptography is the study of techniques in which a message can be transmitted from point A to point B securely, without an eavesdropper being able to discern any of these messages. A classic way to explain cryptography is using one person, Alice, who wants to send a secret message to her friend Bob. Alice wants Bob to be able to understand the message clearly. However, Eve is in the middle of Alice and Bob and has the potential to intercept the message. The point of cryptography is to find ways to encrypt the message so Alice can get the message to Bob without the risk of Eve understanding the message. Then, Bob must be able to decrypt the message in a way that Eve cannot.

A system that allows for this transmission of the message is what we call a cryptosystem. In this system, the message is often referred to as a codeword, although it can be more than one word. The codeword in our normal use of language is referred to as plaintext. An example of a codeword in plaintext is “word.” To make the process more seamless mathematically, the plaintext is translated to numbers in a variety of methods. After being translated into numbers, we can then proceed to further encode the codeword to ensure security before transmitting it across a system. The encrypted codeword is referred to as ciphertext. An example of a codeword in ciphertext is “10110.”

The value of this application of mathematics can be clear when considering the desire and the need to keep information secure. Conversations, letters in the mail, and digital messages (texts, emails, phone calls) are preferably kept secret from any “eavesdropper,” but this becomes even more crucial when sensitive and confidential information is being exchanged. More significant information such as that in bank accounts and national security can be compromised with eavesdroppers. For example, everyday purchases on Amazon require cryptosystems to protect credit card information and other personal user data. Since these transactions are occurring over the Internet, they could be susceptible to hackers. By encrypting the data transmitted, the data is secure against attacks. Therefore, cryptosystems are necessary in keeping people and their information safe from outside attacks.

In an age when quantum computers are on the horizon, however, many existing cryptosystems are not secure to quantum computer attacks. Thus, much of contemporary research in cryptography seeks new methods to maintain security despite the power
quantum computers.

A McEliece cryptosystem introduces errors before transmitting the message, then uncovers the errors as part of the decryption. Our strategy, then, to create a quantum computer-proof cryptosystem will be to randomly introduce errors to the plaintext to create a codeword. We will then use coding theory to correct the errors to decipher the message. Our security relies on Eve (the eavesdropper) not being able to uncover the errors as Bob (the message receiver) can. These errors will be uncovered by Bob using a coding system called uncovering-by-bases using permutations as codewords (see below), in which Bob knows a set of bases that will successfully find and uncover the error. Bob will have this set of bases as part of the private key whereas Eve does not. A base is comparable to a basis from linear algebra and will be defined more thoroughly.

Traditionally and historically, linear codes have been utilized to build cryptosystems. Another way to build a cryptosystem that has been studied less extensively, but has the potential to create a more secure system, is with permutation codes. Permutation codes offer an alternative to transmit messages across a space while maintaining security and efficiency. This method of encryption was introduced in a paper by Srinivasan and Mahalanobis. The authors used permutation codes and 2-sets to translate a codeword up to a larger group, then introduce random errors to the message for additional security. The receiver then uses the uncovering-by-bases to undo any errors, as previously mentioned. However, the security of this system fell short of the necessary security level to work properly against quantum attacks.

With that in mind, we have a slightly revised version of this cryptosystem. We will distance the codeword in the symmetric group by translating up to a larger group size with \((2,k)\)-partitions instead of 2-sets. This came as a result of many other attempts. Ultimately, however, this approach resulted in the most potential, namely with an updated security graph. A foundation in intuition and mathematical concepts, an explanation of both attempts, and results are given in this paper.
Chapter 2

Preliminaries

2.1 Permutation Groups

Definition 2. A group is an algebraic structure consisting of a set of elements and a binary operation that can be applied to any two of those elements. The group must be closed under the operation, have the associative property, contain an identity element, and have an inverse for every element.

Definition 3. A permutation of a set $A$ is a function from $f : A \rightarrow A$ that is 1−1 and onto; a permutation rearranges the elements of $A$.

Definition 4. A permutation group of the set $A$ is a collection of permutations of $A$ that forms a group under composition.

In a permutation group we have two components: the group $G$ of permutations and the set $A$ that is being computed/acted on. The set $A$ could be a subgroup, the group itself, or a non-group (such as 2-sets).

Definition 5. A symmetric group $S_n$ is the group of all permutations of $\{1, 2, ..., n\}$. The number of elements of $S_n$ is $n!$.

Definition 6. The list notation of $S_n$ is a representation of an element that lists the points of an element in the order they result in after being permuted. The length of the element in this form is equal to as many points are in the element ($n!$).

Example 1. The following gives the elements of $S_3$.
The first element fixes all the points, as they all map to themselves:
$1 \mapsto 1$
$2 \mapsto 2$
$3 \mapsto 3$
The second element fixes one point:
1 $\mapsto$ 1
2 $\mapsto$ 3
3 $\mapsto$ 2
Any given element can fix one, two, three, or none of the points.

Example 2. Consider the element 312. This element maps the numbers \{1, 2, 3\} as follows:

1 $\mapsto$ 3
2 $\mapsto$ 1
3 $\mapsto$ 2

Definition 7. Let $G$ be a permutation group such that each element of $G$ permutes the elements in a set $A$. Then $G$ acts on $A$, and we may talk about the action of $G$ on $A$. Then, for $g \in G$ and $a, b \in A$ where $g$ maps $a$ to $b$, then we say $a^g = b$.

Example 3. Let $A = \{1, 2, 3\}$ and $G = S_3$. Take $g \in G$ where $g = 312$ and let $a = 2$. So, consider $a^g = 2^{312} = 1$, as the permutation 312 maps 2 to 1.

Definition 8. A 2-set of $S_n$ is a set of two numbers from the set \{1, 2, ..., $n$\}. For instance, \{1, 2\}, \{3, 4\}, and \{1, 5\} are all 2-sets. A group of permutations can act on these.

Example 4. Actions on 2-sets:
Consider the group $G = S_4$ and the element 3412 $\in G$. Let

$$A = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$$

and take $a = \{1, 2\} \in A$. Then

$$a^g = \{1, 2\}^{3412} = \{3, 4\}$$

Definition 9. A \((2, k)\)-partition is a set of $k$ 2-sets such that the union of the 2-sets is \{1, 2, ..., $n$\}.

Example 5. Actions on \((2, k)\)-partitions:
Consider the group $G = S_6$ and the element 541326 $\in G$. Let

$$A = \{\{\{1, 2\}, \{3, 4\}, \{5, 6\}\}, \{\{1, 2\}, \{3, 5\}, \{4, 6\}\}, \{\{1, 2\}, \{3, 6\}, \{4, 5\}\}, \{\{1, 3\}, \{2, 4\}, \{5, 6\}\}, \{\{1, 5\}, \{2, 6\}, \{3, 4\}\} \ldots$$
and take \( a = \{\{1, 2\}, \{3, 5\}, \{4, 6\}\} \). Then
\[
a^g = \{\{1, 2\}, \{3, 5\}, \{4, 6\}\}^{541326} = \{\{5, 4\}, \{1, 2\}, \{3, 6\}\} = \{\{1, 2\}, \{3, 6\}, \{4, 5\}\}
\]

**Definition 10.** If a group \( G \) acts on a set \( A \), a **point-wise stabiliser** of a subset \( B \) of \( A \) is a subgroup \( H \) such that every element of \( H \) fixes every element of \( B \). For instance, for \( G = S_4 \), the point-wise stabiliser of \( \{1, 3\} \subseteq \{1, 2, 3, 4\} \) is \( H := \{e, 3214, 1432, 3412\} \).

**Definition 11.** A **base** of a permutation group \( G \) acting on a set \( A \) is a subset \( B \) of \( A \) that has the identity as its point-wise stabiliser. This means that given only the action of an element \( g \) of \( G \) on the elements in the base, we can determine exactly how \( g \) acts on all of \( A \).

**Definition 12.** Consider the dihedral group \( D_4 \), which can be thought of as the symmetries of a square.

\[
\begin{array}{ccc}
2 & 3 \\
1 & 4
\end{array}
\]

The permitted actions on this group are four rotation and four reflections. Here are examples of various rotations:

\[
\begin{array}{ccc}
2 & 3 & 1 & 2 & 4 & 1 & 3 & 4 \\
1 & 4 & 4 & 3 & 3 & 2 & 2 & 1
\end{array}
\]

Here is a reflection across the horizontal:

\[
\begin{array}{ccc}
1 & 4 \\
2 & 3
\end{array}
\]

Here is a reflection across the vertical:

\[
\begin{array}{ccc}
3 & 2 \\
4 & 1
\end{array}
\]

Here is a reflection across the diagonal 1/3:

\[
\begin{array}{ccc}
4 & 3 \\
1 & 2
\end{array}
\]
Here is a reflection across the diagonal 2/4:

\[
\begin{array}{cc}
2 & 1 \\
3 & 4 \\
\end{array}
\]

**Example 6.** Consider the non-example of a base for \(D_4\): \(\{2, 4\}\).

\[
\begin{array}{cc}
2 & ? \\
? & 4 \\
\end{array}
\]

If we know where corners 2 and 4 are located in the square, we can have an orientation of: 1, 2, 3, 4 or 3, 2, 1, 4, as follows. Either of these orientations are possible given a diagonal reflection. Thus, given only the positions of 2 and 4, we cannot determine the position of the rest of the elements of the group and so \(\{2, 4\}\) is not a base.

\[
\begin{array}{cc}
2 & 3 \\
1 & 4 \\
\end{array}
\quad \begin{array}{cc}
2 & 1 \\
3 & 4 \\
\end{array}
\]

**Example 7.** An example of a base for this group is \(\{2, 3\}\).

\[
\begin{array}{cc}
2 & 3 \\
? & ? \\
\end{array}
\]

Knowing just where corners 2 and 3 are located, (next to each other), we know that the square must be oriented in the order of corners as such: 1, 2, 3, 4. The only other possibility would be: 4, 2, 3, 1, however this is impossible because there is no reflection or rotation that would yield this positioning of corners. Thus, given \(\{2, 3\}\) we know what the whole positioning of the square must be.

\[
\begin{array}{cc}
2 & 3 \\
1 & 4 \\
\end{array}
\]

There can be one more than one possible base of a group - oftentimes it can be easier to find a larger base to ensure the whole group is covered, but it is more useful to find the smallest possible base. Ideally, we can have a programmed method to find a minimal base of a group, as this process can be long and difficult.
Definition 13. A transposition of a symmetric group is a permutation that interchanges two numbers of an element and leaves all the other numbers fixed.

Example 8. The permutation 2134 is a transposition, as it switches the 1 and the 2, but 3 and 4 are fixed.

Example 9. Finding a base of a group:

Let’s consider the group $S_4$. Elements of this group look like:

\[
\{1234, 1324, 1423, 1432, 2134, 2143, \ldots\}
\]

To find a base, we need elements that, when used together, can determine the position of all elements.

So, let’s add \{1, 2\} to our base. \{1, 2\} tells us about the first two positions, but not about the last two. We add \{1, 3\} to our base and now it is complete. We could keep adding elements and it would still be a base, but taking away either of these would not be possible while keeping it a base.

We denote this as $B = \{\{1, 2\}, \{1, 3\}\}$.

To prove this is sufficient for a base, consider applying every transposition of $S_4$ to this set of bases. The set of transpositions is

\[
\{2134, 3214, 4231, 1324, 1432, 1243\}
\]

\[
\begin{align*}
\{1, 2\}^{2134} & = \{2, 1\} = \{1, 2\} \\
\{1, 3\}^{2134} & = \{2, 3\} \\
\{1, 2\}^{3214} & = \{3, 2\} = \{2, 3\} \\
\{1, 3\}^{3214} & = \{3, 1\} = \{1, 3\} \\
\{1, 2\}^{4231} & = \{4, 2\} = \{2, 4\} \\
\{1, 3\}^{4231} & = \{4, 3\} = \{3, 4\} \\
\{1, 2\}^{1324} & = \{1, 3\} \\
\{1, 3\}^{1324} & = \{1, 2\} \\
\{1, 2\}^{1432} & = \{1, 4\} \\
\{1, 3\}^{1432} & = \{1, 3\} \\
\{1, 2\}^{1243} & = \{1, 2\} \\
\{1, 3\}^{1243} & = \{1, 4\}
\end{align*}
\]

Applying each transposition to each 2-set in $B$ gives us every possible 2-set (there are six of them). So, each 2-set is moved by a transposition on the base $B$.

Example 10. Conjugation of an element in $S_3$ (a group acting on itself where $A = G$):

Consider the element 213 $\in S_3$. Now apply the conjugation $g = 321$. So, we take the conjugation $g$ and permute those elements of 213.

We denote this as $213^g = 213^{321} = 321^{-1} * 213 * 321 = 321 * 213 * 321 = 132$. 

In Python, this process looks slightly different due to the direction in which the elements are read. Above, we read the permutations left to right when computing products, whereas Python interprets elements right to left.

**Definition 14.** The *Hamming distance* of two elements of a permutation group is the number of positions in which they differ. The distance between \( x \) and \( y \) is denoted as \( d(x, y) \).

**Example 11.** Given \( x = 1234 \) and \( y = 1334 \), \( d(x, y) = 1 \), as these elements only differ in position 2.

Given \( x = 12345678 \) and \( y = 98765679 \), \( d(x, y) = 5 \).
Chapter 3

Algebraic Coding Theory

Algebraic coding theory studies the use of error-introduction and error-correction of messages that are transmitted across a network. This is useful in making a message more secure through an added level of encryption, as well as correcting any accidental errors introduced. This ensures that the correct message can be found even with errors added along the way.

Example 12. To understand why error-correction might be useful, consider that we want to transmit the message 123. However, an error could be accidentally introduced by Alice (the sender). So, instead of sending 123, Alice will try to send 111222333, repeating each number three times to ensure the correct message is transmitted. However, an error could be introduced - let 119222333 be the message that is received by Bob. He can correct this one error given the surrounding numbers around the 9, and we still know the message should be 111222333. Thus, we are able to correct an error.

Definition 15. The error-correcting capacity of a group is the maximum number of errors that can be introduced to an element of that group before being sent across a system, so that the errors can be accurately corrected with an uncovering-by-bases.

The following gives us a formula for finding \( r \) for any group \( G \) that is a subgroup of the symmetric group \( S_m \) acting on 2-sets (\( S_m \) will be scaled up to \( S_n \) where \( n = \frac{m(m-1)}{2} \)):

\[
r = m - 3
\]

In the more general case, we have

\[
r = \left\lfloor \frac{m(G) - 1}{2} \right\rfloor
\]

where we define \( m(G) \) as the minimum distance between elements of the group \( G \).

\[
m(G) = n - \max(\text{Fix}(g)) : \forall g \in G
\]

where \( \max(\text{Fix}(g)) \) is the maximum number of fixed points for elements of \( G \).

For instance, in \( S_6 \text{Fix}(312456) = 3 \), as 1, 2, 3 are shifted, but 4, 5, 6 are fixed points.

The derivation of \( r \) for \( S_n \) is derived as follows:
\[ m(G) = n - \max(Fix(g)) = \frac{m(m-1)}{2} - \frac{(m-2)(m-3)}{2} - 1 = \frac{1}{2}(m^2 - m - m^2 + 5m - 6 - 2) = \frac{1}{2}(4m - 8) = 2m - 4 \]

\[ r = \left\lfloor \frac{m(G)-1}{2} \right\rfloor = \left\lfloor \frac{(2m-4)-1}{2} \right\rfloor = \left\lfloor \frac{2m-5}{2} \right\rfloor = \left\lfloor m - 2.5 \right\rfloor = m - 3 \]

**Definition 16.** An uncovering-by-bases (Bailey) is a method to fix \( r \) errors in a permutation group for a given element of the group. An uncovering-by-bases is made up of several bases of the group and given an element which presumably has errors introduced, we iterate through each base and use that base to determine if it could make for a valid element given \( r \) maximum possible errors. This process is continued through all bases of the UBB until an actual element of the group is found with the errors corrected.

**Example 13.** A simple example of what we want to happen is as follows: We have the message 2143 to transmit as a list permutation, but an error is introduced along the way and we transmit 2243 instead. Through UBB we will be able to correct the message back to 2143.

**Example 14.** Performing UBB:

For the sake of this example, we will consider elements of the dihedral group \( D_6 \). Elements of this group work similarly to those of symmetric groups. Each number of this group can be understood as the vertex of a polygon, so \( D_6 \) would be a hexagon with each vertex numbered 1 through 6.

The process of UBB begins on the encryption side with a codeword, which is explained more extensively below in “The Cryptosystem.” For the sake of UBB explanation, assume we have the codeword 123456. After error introduction (in \( D_6 \) we have an error-correcting capacity of 1, so we can introduce 1 error randomly), we have 323456 and this is transmitted across the system.

As a sender, we know the message is 123456, but as a receiver we only know the message is 323456. For the sake of this example we can see what both the sender and receiver know, however, in practice the receiver only knows the message that has an error introduced and we must use UBB to fix the error.

We then must perform UBB to undo the errors and conjugation done. We have \( \{1, 2\} \) as a base and \( \{3, 4\} \) as a base, and together they form an uncovering-by-bases: \( \{\{1, 2\}, \{3, 4\}\} \). We will iterate through these bases.

1. First, consider the base \( \{1, 2\} \). We look at the numbers in these positions in our word - 32 are in positions 1 and 2 of our word.

2. We then iterate through an exhaustive list of the elements of \( D_6 \) until coming upon an element with 32 in positions 1 and 2. In this case, we can find the element 321654.
Elements of $D_6$ in list notation

<table>
<thead>
<tr>
<th></th>
<th>123456</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>612345</td>
</tr>
<tr>
<td></td>
<td>561234</td>
</tr>
<tr>
<td></td>
<td>456123</td>
</tr>
<tr>
<td></td>
<td>345612</td>
</tr>
<tr>
<td></td>
<td>234561</td>
</tr>
<tr>
<td></td>
<td>165432</td>
</tr>
<tr>
<td></td>
<td><strong>321654</strong></td>
</tr>
<tr>
<td></td>
<td>543216</td>
</tr>
<tr>
<td></td>
<td>216543</td>
</tr>
<tr>
<td></td>
<td>432165</td>
</tr>
<tr>
<td></td>
<td>654321</td>
</tr>
</tbody>
</table>

3. Then, take the difference between our word and this found element: $d(323456, 321654) = 3$, as the elements in positions 3, 4, 6 differ.

4. We had previously identified an error-correcting capacity of 1, and seeing as $3 \geq 1$, we must move on to our next base and do another comparison.

5. Now, we consider the base $\{3, 4\}$.

6. Iterating through all elements, we find the element $123456$ which matches the word we received in positions 3 and 4.

7. Then, take the difference between our word and this found element: $d(323456, 123456) = 1$, as only 1 element, that in position 1 differs.

8. Using our error-correcting capacity, we see $1 \leq 1$, so we know that the word $123456$ is the codeword, as we have corrected the number of errors within the limits of the error-correcting capacity.

**Example 15.** Performing UBB (in a larger group):

For this example we will consider elements of the dihedral group $D_8$.
Assume we begin with the codeword 18765432. After error introduction (in $D_8$ we have an error-correcting capacity of 2, so we can introduce 2 errors randomly), we have 18767412 and this is transmitted across the system.
We then must perform UBB to undo the errors and conjugation done. Given an uncovering-by-bases $\{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$, we iterate through these bases.

1. First, consider the base $\{1, 2\}$.

2. We then iterate through an exhaustive list of the elements of $D_8$ until coming upon an element with 18 in the first two positions.
Elements of $D_8$ in cycle notation

<table>
<thead>
<tr>
<th>Elements of $D_8$ in cycle notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
</tr>
<tr>
<td>23456781</td>
</tr>
<tr>
<td>34567812</td>
</tr>
<tr>
<td>45678123</td>
</tr>
<tr>
<td>56781234</td>
</tr>
<tr>
<td>67812345</td>
</tr>
<tr>
<td>78123456</td>
</tr>
<tr>
<td>81234567</td>
</tr>
<tr>
<td>18765432</td>
</tr>
<tr>
<td>21876543</td>
</tr>
<tr>
<td>32187654</td>
</tr>
<tr>
<td>43218765</td>
</tr>
<tr>
<td>54321876</td>
</tr>
<tr>
<td>65432187</td>
</tr>
<tr>
<td>76543218</td>
</tr>
<tr>
<td>87654321</td>
</tr>
</tbody>
</table>

3. Then, take the difference between our word and this found element:

\[ d(17865432, 18765432) = 2, \text{ as the elements in positions 5 and 7 differ.} \]

4. Using our error-correcting capacity, we see $2 \leq 2$, so we know that the word 18765432 is the codeword, as we have corrected the number of errors within the limits of the error-correcting capacity.

It is convenient in this case that our first base of $\{1, 2\}$ did not hold the error. That is, the error was not in positions 1 or 2 and we could find an element of $S_8$ that matches our codeword in those positions. So, we do not need to go on to the other two bases before finding our codeword. If this was not the case, however, we would go onto $\{3, 4\}$ and possibly $\{5, 6\}$, examining the numbers in those positions and finding the difference between the transmitted message until the difference was within the error-correcting capacity.
Chapter 4
The Problem

The foundation of this paper and research question is derived from Srinivasan and Mahalanobis’s work on developing a McEliece cryptosystem with permutation codes. The original work began with a paper published in 2021 and has been further developed using updates in a 2023 version. Acknowledging past literature cryptosystems using linear error-correcting codes, the authors use similar ideas but take advantage of the structure of permutations to transmit messages across a system.

The ideas from this paper found some success, however the system is not secure to ISD (defined below) attacks. That is, the error-correcting capacity is not big enough and the bases are not big enough. Thus, the system is found to be susceptible and messages transmitted not secure enough from attackers.

Example 16. Introducing an error:

Consider obtaining an element from a group $G \leq S_4$. We calculate $r = 4 - 3 = 1$. Given $r = 1$, we can have a maximum of one error introduced to our codeword. Let 3214 be our codeword. Then we can change one number of this element to any other number from 1 to $n$. We randomize and automate this process by iterating once (because $r = 1$), each time selecting a random position of the element to change. Then, a different possible number is randomly selected to replace the current position of the element. The result is a new codeword to transmit across the system that will need to go through UBB to reach it’s original state.

Let 2 be the random position selected. Let 3 be the random number to introduce the error. So, we change the 2 in position 2 to a 3 and end up with a codeword of 3314.

Definition 17. An information set decoding (ISD attack) is a technique used by an eavesdropper or interceptor (known as “Eve” in our introduction) to decrypt an encrypted message. Despite a message being encoded and appearing to be secure, an ISD attack goes through possible bases until a viable base is found. After identifying a base, Eve is closer to being able to decrypt the message. Thus a system with too small of a base is vulnerable to this attack.

Example 17. Consider the group $D_6$ (a hexagon with a number in each corner, actions include rotations and reflections). The error-correction capacity for this group is 1. A UBB
would be \({\{1, 2\}, \{3, 4\}}\), as these are both bases and can correct any error - \(\{1, 2\}\) avoids errors unless in positions 1 and 2, while \(\{3, 4\}\) avoids errors unless in positions 3 and 4. When used together, we can avoid any errors.

Let our message to transmit be 123456, but an error is randomly introduced and instead 623456 is instead transmitted.

Eve, the attacker, does not know about the error, they only know the message 623456 and that the group \(D_6\) is used. The UBB is not known to Eve.

Eve will then iterate through all bases to find a possible solution to correct the error.

Eve will start going through the bases:

\[
\begin{align*}
\{1, 2\} \\
\{1, 3\} \\
\{1, 4\} \\
\{1, 5\} \\
\{1, 6\} \\
\{2, 3\}
\end{align*}
\]

Eve will first check \(\{1, 2\}\) and find that there is no possible element of \(D_6\) matching 62\underline{3456}, and Eve will find failure until reaching \(\{2, 3\}\) at which point the element 62\underline{3456} is identified as a possible match and can correct the error to find the message 123456.

Note that Eve can choose the order of bases she tries in any order. So, it may take her many failures and tries to reach a base that will work to uncover, or she could guess a correct base on the first try. The best case scenario is trying one base, the worst case scenario is trying all the possible bases.

Thus, Eve can complete a successful ISD attack and although it takes longer than Bob, who would know to use the UBB \(\{{\{1, 2\}, \{3, 4\}}\}\), Eve will eventually find the base that works.

So, to combat this type of attack, we need it to be impossible or infeasible for Eve to uncover the errors. There there needs to be:

1. More error-correction capacity
2. Longer bases

Both of these allow for a higher probability that no matter the base Eve uses to try and uncover the errors, there will still be an error that intersects this position. Thus, Eve wouldn’t be able to be able to find a valid element from the errors she attempts to uncover.

This use of permutation codes instead of linear-codes, as secure cryptosystems have done, has not yet proven successful, yet the process to do so is valuable and insightful for future work.
Chapter 5

Srinivasan and Mahalanobis’s Cryptosystem

An attempt at a cryptosystem using permutation codes with 2-sets is explained here. This is based off of the paper by Srinivasan and Mahalanobis. Although unsuccessful, understanding the process gives way to revisions for different versions, possibly with more success.

1. Generate a codeword: select a group to generate a codeword from (e.g., $S_4$, $S_6$, $S_{20}$). Given the group $S_m$, there are $m!$ possible codewords from which we can choose. This step is randomized with an arbitrary selection of an element of the group. We call our codeword $c$.

2. Translate the codeword to a larger group: taking our codeword that is an element in $S_m$, we translate up to a larger group using 2-sets of elements of the group $S_m$. In particular, we move into $S_n$ where $n = \frac{m!}{2(m-2)!}$, also known as $\binom{m}{2}$, $m$ choose 2. To do this, we generate all 2-sets in $S_m$, giving each 2-set a numerical labeling so that each 2-set in $S_m$ is numbered from 1 to $n$. The labelings can be generated randomly, in which case step 3) is not necessary because the conjugation/scrambling happens here. If the labelings are assigned numerically, step 3) is still included. These labelings are a conjugation which we will call $g$.

Once we have our labelings of each 2-set, we take each number of the codeword and translate it using the labelings, applying the calculation $gc$.

Consider the following example of an element from $S_4$ being translated to $S_6$. $c = 2143$ and we have the following labelings:

1 = \{1, 4\}
2 = \{1, 2\}
3 = \{3, 4\}
4 = \{1, 3\}
5 = \{2, 4\}
6 = \{2, 3\}
Go through the labelings. Given $c = 2143$,

\[
\begin{align*}
1 &= \{1, 4\} \mapsto \{2, 3\} = 6 \\
2 &= \{1, 2\} \mapsto \{2, 1\} = 2 \\
3 &= \{3, 4\} \mapsto \{4, 3\} = 3 \\
4 &= \{1, 3\} \mapsto \{2, 4\} = 5 \\
5 &= \{2, 4\} \mapsto \{1, 3\} = 4 \\
6 &= \{2, 3\} \mapsto \{1, 4\} = 1
\end{align*}
\]

Thus, we have a new element of $623541$ by applying the codeword $c$ to the labelings $g$. The purpose of this step is to increase the size of the possibilities for codewords while maintaining our codeword, just disguised in a larger group. Additionally, this step increases the space between group elements, allowing error-correction to work.

3. Conjugate the translated codeword: to further scramble the codeword, a conjugation is applied to the codeword. This conjugation is a randomly generated element of the group $S_m$.

To further our example above, the conjugation $g = 463125$ could be applied, and note again that $623541 \cong 623541$. So, $623541 = 623541^g = 623541^{463125} \mapsto 643251$.

4. Introduce error(s): given an error-correcting capacity of $r$, randomly introduce $r$ errors into the codeword. Iterate through the element and make those random changes by altering the number at a given position.

5. Transmit message

6. Perform UBB: now we are on the other side of the cryptosystem, having received the transmitted codeword. To correct the errors introduced before transmission, uncovering-by-bases is performed. See above for a deeper explanation of UBB.

7. Translate down to smaller group size: this final step is the reverse of step (2). Again, we use a dictionary of 2-sets that correspond to a number in $S_m$. Each number in the codeword being translated down maps to a number in the smaller group, back down to $S_n$, resulting in a codeword back in $S_n$ where the encoder began.

The following is a graph representation of the security level as a function of the size of the group. The formulas used here and the graph itself are derived from Srinivasan’s and Mahalanobis’s paper, as is the method explained up until this point.

The $x$-axis is the size of our group ($n$) and the $y$-axis is the security calculated from the following formulas:

\[
H_2(\frac{x}{n}) - (1 - \frac{k}{n})H_2(\frac{r}{n-k})
\]

\[
H_2(x) = -x \log_2(x) - (1 - x) \log_2(1 - x)
\]
\( n \) is the size of the group: \( \frac{x(x-1)}{2} \)
\( r \) is the error-correcting capacity: \( x - 3 \)
\( k \) is the maximal base size: \( x \cdot \log_2(x) \)

The above graph shows a ceiling on how high the security gets, leveling off between 20 and 25 even as \( n \) grows very large. Ideally, the security level would near 80-bits. This means we need to find a new method with improved security.
Chapter 6

Our Cryptosystem

We attempted many more than two attempts, but this is the second attempt genuinely worth sharing extensively. However, it took many trials and failures to get here. What follows is my contribution to the research, outlining a new and promising cryptosystem with improved security compared to the 2-sets approach from above.

The process outlined above yielded to be unsuccessful primarily due to a lack of proper security of the system. This lack of security stemmed from a base that is not large enough as well as an ability to correct more errors. As is, the larger the group, the error-correcting capacity trended towards zero which is problematic.

The following change can be made to the cryptosystem in an attempt to solve the issues presented:

1. We will use a partition of 2-sets instead of singular 2-sets as an element of the group.
   Follow the same process as above, however, instead of translating with 2-set labelings in step 2), \((2,k)\)-partitions are used instead.
   In example, using \(S_6\) we could have:

\[
\begin{align*}
\mathcal{I} &= \{\{1,2\}, \{3,4\}, \{5,6\}\} \\
\mathcal{I}_2 &= \{\{1,2\}, \{3,5\}, \{4,6\}\} \\
\mathcal{I}_3 &= \{\{1,2\}, \{3,6\}, \{4,5\}\} \\
\mathcal{I}_4 &= \{\{1,3\}, \{2,4\}, \{5,6\}\} \\
\mathcal{I}_5 &= \{\{1,3\}, \{2,5\}, \{3,6\}\} \\
\mathcal{I}_6 &= \{\{1,3\}, \{2,6\}, \{4,5\}\} \\
\mathcal{I}_7 &= \{\{1,4\}, \{2,3\}, \{5,6\}\} \\
\mathcal{I}_8 &= \{\{1,4\}, \{2,5\}, \{3,6\}\} \\
\mathcal{I}_9 &= \{\{1,4\}, \{2,6\}, \{3,5\}\} \\
\mathcal{I}_{10} &= \{\{1,5\}, \{2,3\}, \{4,6\}\} \\
\mathcal{I}_{11} &= \{\{1,5\}, \{2,4\}, \{3,6\}\} \\
\mathcal{I}_{12} &= \{\{1,5\}, \{2,6\}, \{3,4\}\} \\
\mathcal{I}_{13} &= \{\{1,6\}, \{2,3\}, \{4,5\}\} \\
\mathcal{I}_{14} &= \{\{1,6\}, \{2,4\}, \{3,5\}\} \\
\mathcal{I}_{15} &= \{\{1,6\}, \{2,5\}, \{3,4\}\}
\end{align*}
\]

Using these partitions instead of 2-sets allows for the codeword to be translated up to a
larger group, in turn allowing for larger error-correction which is important for better security. In particular, we can see that, for 2-sets, we have a size of $\frac{m(m-1)}{2}$ to expand to, whereas our newer partition approach expands to $(m - 1)!!$.

**Definition 18.** The double factorial is defined as follows:

$$m!! = \begin{cases} 
m * (m-2) * (m-4) * ... * 5 * 3 * 1 & : m > 0 \text{ odd} \\
m * (m-2) * (m-4) * ... * 6 * 4 * 2 & : m > 0 \text{ even} \\
1 & : m = -1, 0
\end{cases}$$

**Example 18.** Here are a couple examples of how to use the double factorial.

$$5!! = 5 * 3 * 1 = 15$$

$$(m - 1)!! = (m - 1) * (m - 3) * (m - 5) * ... * (m - (m - 1))$$

**Example 19.** The double factorial is derived by considering how to determine the possible numbers that can go into the positions of a $(2, k)$-partition. To understand this, take the $(2, k)$-partitions in $S_6$. We start with fixing a single number, let’s say 1 in the first position: $\{1, _, _, _, _, \}$. There are five possibilities for the next position, let’s say we choose 2: $\{1, 2, _, _, _, \}$. Then we fix a single number in the next position, let’s say 3: $\{1, 2, 3, _, _, \}$. There are three possibilities for the next position, let’s say we choose 4: $\{1, 2, 3, 4, _, \}$. Then we fix a single number in the next position, let’s say 5: $\{1, 2, 3, 4, 5, \}$. There is one possibility for the final position: $\{1, 2, 3, 4, 5, 6\}$. Thus, we use the number of possibilities in each position and multiplying them together gives the number of possible combinations of orderings. For this example we have $5 * 3 * 1 = 5!! = (6 - 1)!!$. This can be generalized to the size of the group $m$, which is how we have derived $(m - 1)!!$.

We can analyze these numbers in each of the methods in the following table.

<table>
<thead>
<tr>
<th>$m$</th>
<th>2-sets $\frac{m(m-1)}{2}$</th>
<th>$(m - 1)!!$</th>
<th>$(m - 1)!!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>105</td>
<td></td>
</tr>
</tbody>
</table>

As is evident from the above table, the partition approach allows for a much faster expansion of group size expansion once we exceed a permutation degree of 6. In practice, we want to use very large group sizes, so as $m$ increases, the size of the expanded group gets larger and larger with bigger improvements compared to the 2-sets approach.
Using the revised security for \((2, k)\)-partitions as opposed to 2-sets, I’ve attempted a graph similar to that for the 2-sets, hoping for a higher security level ceiling. The following is a graph representation of the security level as a function of the size of the group once again. The security level and \(H_2\) formulas are the same as the 2-set example, as given in Srinivasan and Mahalanobis’ paper. The \(n\) and \(r\) values are different - however, we’ve used the same \(k\) value as the 2-set approach. This is an uncertain decision and is part of an open question for improved accuracy of security estimates.

\(n\) is the size of the group: \((x - 1)!!\)

\(r\) is the error-correcting capacity: \(\frac{(x-2)(x-3)!!-1}{2}\)

\(k\) is the maximal base size: \(x \times \log_2(x)\)

You can see the graph reaches a security of around 400-bits, rising above our desired threshold of 80-bits around a group size of 20. The graph also takes a dramatic drop down to zero at a group size of around 30 – 35. This could be due to the double factorial getting so large that the limits of the computations of graphing are reached. However, this is also part of an open question for further exploration.

Despite the uncertainties of this graph, the high security the graph reaches provide hope that this new \((2, k)\)-partitions approach is better than the 2-sets approach in terms of improved security.
Chapter 7

Programming the Process

The problem in the scale demonstrated in the examples here are not conducive to a secure cryptosystem. However, being able to scale the problem to larger groups and perform the process multiple times one after another would benefit the attempt to find a fast and secure cryptosystem. Thus, a Python version of this process has been started and built with a loose structure. This set of code has a long way to go still, given some specifics are not present in the program and it has yet to be tested.

Future work on the Python code utilizing the math presented here would give way to a productive research project to eventually have a fully functioning cryptosystem in Python. The program uses SymPy, a package of Python that includes Permutation, PermutationGroup, and SymmetricGroup symbols that function the way we need.

The program as it is has several functions:

1. makeSets: generates the 2-sets of a symmetric group of a given size; since these are not already stored anywhere in this library, we generate them here.

2. labelSets: each 2-set is given a labeling. Python’s dictionary works well for this. Although the 2-sets are created in numerical order, we label them randomly so as to add an additional layer of security. This serves the same purpose as labeling the 2-sets numerically and then conjugating (scrambling) them with a permutation element.

3. cycleToList: translates a permutation from cycle form to list form.

4. listToCycle: translates a permutation from list form to cycle form.

5. translateUp: scales an element/permutation of the Symmetric group up to a larger group. Given an element in Symmetric group size $n$, the element returned will be in Symmetric group size $\frac{n!}{2(n-2)!}$. This function has yet to be tested out.

6. addErrors: given an error-correcting capacity, we introduce random errors to the given element and return a new element with errors to be transmitted across the system.
To be still written and get working:

1. UBB: perform an uncovering-by-bases on a given message that has some error(s) introduced. A simple UBB algorithm is applied here.

2. translateDown: just as translateUp scales an element/permutation to a larger group, this will be able to translate that element/permutation back down to a smaller group size that it came from. This is important in the decoding process.
import sympy as sp
from sympy.combinatorics import Permutation, PermutationGroup
from sympy.combinatorics.named_groups import SymmetricGroup
import random
from math import factorial

S4 = SymmetricGroup(4)
B = S4.base
n = 4
k = 2 #for 2-sets

def makeSets(n):
    #n = factorial(n)/(factorial(k) * factorial(n-k))
    sets = {}
    i = 0
    for j in range(n-1):
        for l in range(n-1, j, -1):
            sets[i] = Permutation(j, l)
            i+=1

    return sets

"""
file to represent permutation groups, using sympy
"""
```
def labelSets(sets):
    twoSets = {}
    indices = []
    for i in range(len(sets)):
        indices.append(i)
    random.shuffle(indices)

    for i, j in zip(indices, range(len(indices))):
        twoSets[j] = dict[i]
    return twoSets

makeG(twoSets):
    G = PermutationGroup()
    for i in range(len(twoSets)):
        G.append(twoSets[i])
    return G

translatePerm(n, k, e, twoSets):
    newPerm = []
m = factorial(n)/(factorial(k) * factorial(n-k))
#for i in range(factorial(n)): #iterate through to get each element of the group [24]
    keys = list(twoSets.keys())
```python
values = list(twoSets.values())

for j in range(m): #iterate through the ‘‘underlined” labelings/k–sets to translate each element [6]
    p = Permutation([twoSets[j]])
    position = values.index(e(p))
    newPerm.append(keys[position])

return newPerm

"""
def translateUp(n, k, e, twoSets):
    list = []

    for i in range(factorial(n)/(factorial(k) * factorial(n-k))):
        val = twoSets[i]
        valNew = e(val) #val(e)??
        iNew = twoSets.get(valNew)
        list.append(iNew)

    return Permutation(list)

"""
translate cycle notation to list notation for permutations
@param: cycle, the permutation in cycle notation
@return: list, the permutation in list notation
"""
def cycleToList(cycle):
    p = Permutation([cycle])
    makeList = []
    for i in cycle.length():
        makeList.append(i)
    return p(makeList)

"""
translate list notation to cycle notation for permutations
@param: list, the permutation in list notation
@return: cycle, the permutation in cycle notation
"""
def listToCycle(list):
    return Permutation([list])

"""
```
introduce errors to a permutation/element
@param: r, error-correcting capacity of this size of group
(max. number of errors that can be corrected)
@param: e, the element to introduce errors to
@param: n, size of the Symmetric group we’re working in
@return: eNew, the element with errors introduced

```python
def addErrors(r, e, n):
    list = cycleToList(e)
    for i in range(r):
        ind = e.index(random.randint(0, n-1))
        list[ind] = random.randint(0, n-1)
    return list
```
Chapter 9

References


