INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A

Properties of Shuffle Groups and their Relationship to Cryptography

by

Daniella Bak

A dissertation submitted to the Graduate Faculty in Mathematics

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy, The City University of New York

2000

UMI Number: 9986301

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

This manuscript has been read and accepted for the Graduate Faculty in Mathematics in satisfaction of the dissertation requirement for the Degree of Philosophy.

9/14/2005
Date

Michael Anshel
Chair of Examining Committee

9/15/00
Date

Executive Officer

Michael Anshel

Burton Randol

Alphonse Vasquez

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

PROPERTIES OF SHUFFLE GROUPS

AND THEIR RELATIONSHIP TO CRYPTOGRAPHY

by

Daniella Bak

Advisor: Professor Michael Anshel

This paper deals with three applications of group theory to cryptography. Two involve methods of public-key encryption with a private key for decryption. The first, proposed in [Wag84], uses the Word Problem, while the second method, proposed in [Ansh93], makes use of the conjugacy problem. In both cases, the methods assume that the underlying group is infinite.

We consider the possibility of applying these methods using finite groups. While we obtain some results applicable to all finite groups, we focus specifically on the Shuffle Group. This group, which originated in an investigation of popular magic tricks using certain types of "perfect shuffles", has been thoroughly investigated in a classical paper [Dia83]. In brief, it was found that the underlying group structure breaks down into five categories, depending on the number of cards in the underlying deck. In our results, we found that four of the five types, where $2n \neq 2^k$, could be analyzed using the same methods, while the fifth case, where the deck is of size $2n = 2^k$, required a

different approach.

We found that the Word Problem could be solved in the first case with an order of magnitude that is linear in the size of the input, and of much smaller magnitude than the order of the group. While the method presented could apply equally to the second case, it would not be as efficient. Instead, we offer a second method, using a canonical representation for the group elements.

The Conjugacy Problem, in both cases, uses the solution to the Word Problem. An analysis of the cycle structures of permutations in the Shuffle Group yields an efficient method for solving the Conjugacy Problem when $2n \neq 2^k$. This method cannot be applied in the case where $2n = 2^k$, but again an alternative efficient method using the canonical representation is presented.

The third application of group theory is to a protocol for a secret-key exchange. This application seems most likely to be effective with finite groups. However, several pitfalls must be avoided in this case. We make several observations about the avoidance of these pitfalls, specifically with regard to the Shuffle Group.

Acknowledgments

I am extremely grateful to my advisor, Dr. Michael Anshel, who always believed that I would finish, even when I was sure I wouldn't. He was more than encouraging, and gave me more credit than I deserved; when I called month after month and reported no progress, he sympathized with the workload that I had, and assured me that I would accomplish more in the near future. His insights into cryptography and group theory inspired this paper.

To my father, who sparked my interest in mathematics and computer science at an early age. He was the reason for my popularity among the students in my calculus class, who knew that I was a phone call away from the best explanations possible. When I thought I could not possibly learn all of Abstract Algebra over one winter break, he forced me to sit and study, and talked me through Sylow Theorems. And now, at the end of this leg of the journey, once again he has been there all summer to edit, advise and correct my misconceptions.

To my mother, who claims that she gets pleasure from doing my laundry and making chicken soup for me and anyone in my apartment. She taught me that I could get a doctorate and still enjoy the summer by the pool.

To Dr. Burton Randol and Dr. Alphonse Vasquez, who inspired me in their classes during my first year in graduate school, and who sat on both my orals and my dissertation defense committees. To Noson, who sat through my entire orals presentation a week before my committee did, and who once again gave me many useful suggestions. To Avi, who spent hours getting Latex to work on my laptop, and who's never been acknowledged in a thesis before.

Finally, to the friends who keep asking, "Are you done yet?"; I can finally say yes.

To Aba, with love and gratitude.

Contents

List of Figures

Chapter 1

Introduction

1.1 Importance of Cryptography at the Beginning of the 21st century

It is hard to ignore the rise in popularity of cryptography in the last few decades. Anyone who has typed in a credit card number on-line has thought about security. Many internet sites recommend the use of a secure server when transmitting confidential information. Even casual users of Netscape Navigator are aware of the little key icon on the bottom left of the screen, indicating whether or not the information being transmitted has been protected with encryption. With the rise of communication via email, cellular phones and the like, encryption schemes will continue to play a large role in

our lives.

The NSA, or National Security Agency, is the official communications security body of the U.S. government. It was given its charter by President Truman in the early 50's, and has continued research in cryptology until the present. (*Note: cryptology* is the study of encryption, as opposed to *cryptography*, the writing of codes). The NSA is known to be the largest employer of mathematicians in the world, and is also the largest purchaser of computer hardware in the world. For many years, the NSA was very protective of its cryptographic knowledge. Very little information was disseminated, and there are rumors that research in the area not under the auspices of the NSA was discouraged. Especially in the years after World War II, information was considered classified and was very hard to come by. However, in the past few years, research in cryptography is booming, and many books, papers and journals are devoted to the field. The government still tries to monitor the field, by imposing restrictions on the complexity of cryptographic schemes which can be exported to other countries, although the restrictions in this area have been greatly reduced in recent years.

A major concern in the area of cryptography is that with processing speeds of computers growing so quickly, schemes that were once adequate are

quickly becoming obsolete. The much publicized RSA-129 (Rivest-Shamir-Adelman algorithm, based on a 129 bit key), was broken in 1994 using 1600 computers. Versions of RSA in use today are based on larger keys. The need for schemes based on other mathematical problems is great.

1.2 Cryptography Concepts

Cryptography is used to send messages from a *sender* to a *receiver*, in a form that is, for practical purposes, undecipherable to anyone who eavesdrops along the way. The original message is called *plaintext*; the sender will *encrypt* the plaintext and then send it to the receiver, who will subsequently *decrypt* it.

In classical cryptography, one key is selected by both A and B (the sender and the receiver), usually in a face-to-face meeting, and it is kept secret. The encryption key can either be calculated from the decryption key, or the encryption and decryption keys can be the same. The problem with this system is that if the key becomes known, the system becomes insecure, and anyone can encrypt or decrypt messages. Additionally, since an extra measure of security is required in order to transmit the private key, changing the key may present further difficulties.

A different system is known as *Public Key Cryptography*, so called because in this scheme, the encryption key is public. There is a corresponding decryption key known only to the person decrypting the message. This scheme is described in [Stin95], as a box with an open combination lock. The encrypter can place a message within the box and then close the box with the combination lock. Once it is locked, however, the only person who can open the box is the one with the combination (the decrypter).

How can a public key, a key known to everyone, remain secure? The answer is that decryption cannot be feasibly performed without knowledge of the private key. However, the private key does not have to be transmitted from the sender to the receiver, and therefore remains secure.

In any public-key cryptographic scheme it is essential to include a 'trapdoor function', with the property that computation in one direction (encryption) is easy and in the other (decryption), it is virtually impossible without private knowledge, i.e. the trapdoor has the special property that the reversal of the computation is reasonable if the private key K is known. These functions, known as *one-way functions*, are believed to include factoring, since multiplying two large primes together is a simple mathematical calculation, but factoring a large composite number into two prime multiples is thought

to be hard. There IS a brute-force algorithm, but it takes much more time in this direction. This example is the basis of the RSA cryptographic scheme, as well as many others.

1.3 Complexity

In general, cryptographic schemes are not considered to be unbreakable; rather, they are considered to be unbreakable within a given amount of time. A problem that is solvable in polynomial time, based on the input, is said to be "in P". This designation is system-independent; it depends solely on the size of the input. For example, if an algorithm can decrypt a message of length n in n^2 steps, or n^3 steps, then it is said to be $O(n^2)$ or $O(n^3)$ respectively, each of which is a polynomial-time algorithm. However, if the algorithm takes 2^n steps, then it is considered to be exponential, which makes it slower than any polynomial-time algorithm.

Many public-key cryptographic schemes are based on problems that are considered to be in NP, in other words, problems which are **thought** to have no solution in polynomial time, but whose solutions can be verified in polynomial time. RSA is based on factoring, a problem that is NP-Complete. (NP-Complete problems are problems in NP, such that if they *are* found to be

solvable within polynomial time, then the much studied result P=NP would turn out to be true.)

Much work has gone into analyzing the complexity of factoring. There are many problems that are considered to be NP-Complete, whose complexities have been analyzed. One such problem is Dehn's Conjugacy Problem, a problem in the area of Group Theory, which will be elaborated upon in the next section.

Chapter 2

Group Theory Concepts

2.1 Generators and Relations

[Cox65] The elements $S_1, S_2, ..., S_m$ of a discrete group G are called a set of *generators* if every element of G can be expressed as a finite product of their positive and negative powers. A set of relations $g_k(S_1, ..., S_m) = Id$ is called a *presentation* of G if every relation satisfied by the generators is an algebraic consequence of these particular relations.

The notation for a group defined by a set of generators and relations will be

$$G =< S_1, S_2, ..., S_m | R_1, R_2, ..., R_k = Id >$$

2.2 The Word Problem and The Conjugacy Problem

Given a unique set of generators $a, b, c...$ and a unique (possibly empty) set of relations (P, Q, R, etc.), there is a unique group G (up to isomorphism) [Mag66] with the presentation

$$G =< a, b, c, ...; P, Q, R, ... >$$

Regarding groups presented in such a fashion, Max Dehn, in 1911, formulated what he called "three fundamental decision problem". They are as follows:

1. The Word Problem

   For an arbitrary word W in the generators, decide in a finite number of steps whether W defines the identity element of G or not. (Note: This problem can be restated as the problem of whether or not an arbitrary word W_1 is the same as another arbitrary word W_2).

2. The Conjugacy Problem

   For two arbitrary words W_1 and W_2 in the generators, decide in a finite number of steps whether or not W_1 and W_2 define conjugate elements of G.

3. The Isomorphism Problem

For an arbitrary group G' defined by means of another presentation, decide in a finite number of steps whether G is isomorphic to G'.

The Word Problem was proven to be unsolvable for some finitely presented groups. [Mag66] has a solution to the word problem for one-relator groups and some two-relator groups. There is currently a debate about the conjugacy problem for two-relator groups; there is a published solution, but there is still a question about its accuracy.

In this paper we will be focusing on variations of the Word Problem and the Conjugacy Problem. That is, we will consider large finite groups and seek to determine whether two expressions in the group generators can be identified as representing the same or conjugate elements in polynomial time. There has not been much said about either the word or the conjugacy problem for finite groups. We will be dealing specifically with *The Shuffle Group*, which will be discussed in further detail in a later section.

2.3 Golomb's Algorithm

A very simple and straightforward technique for characterizing conjugates in a group is discussed in [Gol96]. We will provide a brief description of this

method. A pseudocode algorithm as well as code for this method appear in the appendix.

One way to determine whether a and b are conjugates in a finite group is to simply calculate gag^{-1} for all g in G, and see whether any of these terms equals b. However, for some problems this method is computationally unfeasible. In the world of cryptography, where solutions are measured by the time that they take, this method is simply unusable.

Golomb noticed a remarkably simple test of conjugacy. That is, two elements a and b of a finite group are conjugate if and only if they appear in symmetric locations across the diagonal of the group table. In other words, a and b are conjugate if and only if there exist elements u,v such that $a = uv$ and $b = vu$.

The proof is immediate. On the one hand, uv and vu are conjugates, since $uv = u(vu)u^{-1}$. On the other hand, if $a = cbc^{-1}$, then, with $u = c$, and $v = bc^{-1}$, $a = uv$ and $b = vu$.

We will consider the efficiency of Golomb's Algorithm for determining conjugates.

Chapter 3

The Design and Analysis of a Combinatorial Cryptosystem

3.1 Definition of Terms

As mentioned above, many public-key cryptographic schemes are based on problems which are considered to have no polynomial time solution. The focus of this paper will be to outline the elements of a cryptographic scheme based on Dehn's conjugacy problem and the shuffle group. The ideas in this section stem from [Ansh93].

3.2 The Basics

A public-key cryptosystem based on the algorithmic unsolvability of the word problem was suggested by N.R. Wagner and M.R. Magyarik. The basic idea of their system is as follows:

Given a finitely presented group G with algorithmically unsolvable word problem, let two words y_0 and y_1 from the group represent the bits 0 and 1 in the message to be transmitted. The message is encoded by sending, in place of the original words y_0 and y_1, various words in the group generators which represent the two elements. Thus, all words equivalent to y_0 will represent 0, and all words equivalent to y_1 will represent 1. Since our underlying group has an algorithmically unsolvable word problem, our message is secure.

In order for the recipient to decode the message, he is given a secret homomorphism

$$h : G \rightarrow A$$

to a finitely presented group A with efficiently solvable word problem (possibly even a large finite group).

[Ansh93] extended the work of [Wag84] by proposing the use of a finitely presented, residually finite group with algorithmically unsolvable conjugacy problem. For each element w in a residually finite group G, $\exists$ a subgroup

N_w such that w is not an element of N_w. For such a group, let z represent the identity and w represent any other element of the group. Since G was chosen to be residually finite there exists a finite image of G, G/N_w such that w does not belong to N_w.

Then when we consider the homomorphism

$$h : G \to G/N_w$$

it follows that

$$h(w) \neq 1,\ h(z) = 1$$

Hence we conclude that w and z and $h(w)$ and $h(z)$ are non-conjugate pairs in G and G/N_w, respectively. We keep the homomorphism h secret and assume that computation in the finite group is efficient enough to determine when two elements specified by words are conjugate or whether a word defines the identity in G/N_w. The mechanism for encoding now follows that of the word problem cryptosystem described above with the enhancement of conjugation by words in G (as well as insertions and deletions of defining relators) being allowed. That is, all expressions equivalent not only to z but to all conjugates of z represent 1, while all expressions equivalent to conjugates of w could represent 0.

3.3 Use of the Shuffle Group in a Cryptosystem

Although the cryptosystem described above assumed the use of an infinite group, the question has been raised about the possible use of a finite group. In particular, we will examine aspects of the Shuffle Group to determine its suitability. In brief, we have discovered that it would not serve well as a replacement for the infinite group with unsolvable word or conjugacy problem, since both the word and conjugacy problems have very elegant and quick solutions in the Shuffle Group. On the other hand, this solvability of the word and conjugacy problems might make it a suitable candidate for the image of the homomorphism described above.

3.4 A Group-Theoretic Key Establishment Protocol

The existence of a private key raises the related question of how the private key is to be transmitted securely from the encrypter to the decrypter. Here, too, [Ansh99] suggests the use of group-theoretic ideas, specifically commutators.

The commutator, $XYX^{-1}Y^{-1}$, can be viewed two ways: as the product of a conjugate of Y with Y^{-1}, or as the product of X with a conjugate of X^{-1}.

In [Ansh99], this idea is exploited for a key-establishment protocol based on the Braid Group, a group with an unsolvable conjugacy problem. In this paper we investigate the feasibility of using the Shuffle Group, a finite group, for a similar purpose. First, we enumerate the protocol as described in [Ansh99]:

3.4.1 The Protocol

The parties Alice and Bob publish, respectively, words

$$a(1),a(2),...,a(n)$$

$$b(1),b(2),...,b(m)$$

specifying group elements from a group $G = \{S|D\}$ with generating symbols S and defining relations D. The common key K is created by the following concurrent actions of Alice and Bob.

3.4.2 Alice's Actions:

(1a) Alice selects a private word X in the subgroup of G generated by $a(1)$, $a(2)$, ..., $a(n)$ and conjugates each $b(i)$ by that X producing $b^*(1), b^*(2), ..., b^*(m)$

(2a) Alice selects a word $B(i)$ in the generating symbols such that $B(i)$ and $b^*(i)$ define the same element of G for $i = 1, ..., m$.

In other words, $B(i)$ is a disguised form of $b^*(i)$ so that knowing $B(i)$ does not obviously reveal what X is.

(3a) Alice transmits to Bob

$$B(1),\ B(2)\ ,...,\ B(m)$$

Bob (as well as anyone else listening in) knows the values of his chosen elements $b(1), ..., b(m)$, as well as the conjugates $B(1), ..., B(m)$, but is unaware of X.

3.4.3 Bob's Actions:

Bob transforms Alice's public words in a similar manner:

(1b) Bob selects a private word Y in the subgroup of G generated by $b(1), b(2), ..., b(m)$ and conjugates each $a(i)$ by that Y producing $a^*(1), a^*(2), ..., a^*(n)$

(2b) Bob selects a word $A(i)$ in the generating symbols such that $A(i)$ and $a^*(i)$ define the same element of G for $i = 1, ..., n$

(3b) Bob transmits to Alice

$$A(1), A(2), ..., A(n)$$

Alice (as well as any eavesdropper) knows the values of her $a(i)$'s, as well as the Y-conjugates of them, but she does not know the value of Y.

3.4.4 Common Actions:

(4ab) Alice and Bob now compute, respectively, V and W, each of which specifies the commutator $C = [X, Y]$ defined by the words X and Y.

Alice determines the commutator in the form $X * (YXY^{-1})^{-1}$. She finds YXY^{-1} as follows:

Since X is a function of $a(1), ..., a(n)$, e.g. $X = a_1 a_2^2 a_3$, and since Alice is in possession of the conjugates, she can produce the same function of the conjugates, obtaining YXY^{-1}=$A_1 A_2^2 A_3$ [= $y a_1 y^{-1} (y a_2 y^{-1})^2 y a_3 y^{-1}$]

(5ab) Alice and Bob compute a common key $K = F(V) = F(W)$, where F is a one-way function from the words in the generating symbols to words in the binary alphabet 0,1 such that $F(W) = F(Z)$ if and only if W and Z define the same element of G.

3.4.5 The Braid-Group Platform

By a platform for a protocol, we mean a choice of group G to support both security and performance requirements of a cryptographic system. One such group, the braid Group B_n is given by:

$$< s_1, s_2, ..., s_n | s_i s_{i+1} s_i = s_{i+1} s_i s_{i+1}, s_j s_k = s_k s_j,$$

$$1 \leq i, j, k < n \text{ and } |j - k| \geq 2 >$$

Aside from the word and conjugacy problems, then, we will deal with the suitability of the Shuffle Group for this key-exchange protocol. Here we are dealing with a different type of conjugacy problem. In the classic conjugacy problem, one is given two elements and challenged to determine whether or not they are conjugates. Here we are given at least two elements, $a_1, ..., a_n$, along with their conjugates $A_1, ..., A_n$, where $A_i = y a_i y^{-1}$ for each i=1,2,...,n. The question is whether y can be determined from the above.

Chapter 4

The Shuffle Group

4.1 Introduction

The concept of a *perfect shuffle* has been studied both by mathematicians and by magicians. A perfect shuffle is a specific type of permutation of a set of numbers, (or more specifically for magicians, a deck of cards). The trick is to split the deck in half, and then perfectly interleave the two halves with each other. Perfect shuffles are interesting to magicians because of properties which lend themselves to many magic tricks. For example, if one was to perform 8 perfect shuffles on a deck of 52 cards, the deck would return to its original order. In addition, any deck with 2^k cards can be reversed in k perfect shuffles. There is also an algorithm in [Ram96] which describes

how to move any card in the deck to the top of the deck. Many papers have been written about these concepts. A book written by a mathematician and a magician [Morr98] is devoted entirely to perfect shuffles. Each chapter begins with a description of a magic trick that uses perfect shuffles, and then the chapter proceeds to explain the mathematical concepts involved in the trick. As an aside, this book also includes, in an appendix, a method for actually perfectly shuffling a deck of cards. As I've discovered first-hand, this is not a simple physical task.

4.2 Mathematical Details

Suppose you have a deck with an even number of cards. There are actually two ways to "perfect shuffle" this deck. In the first way, which we will call an *Out Shuffle*, the cards are interleaved by taking the top card from the first pile, followed by the top card from the second pile, and placing them face down in a new pile. Continue alternating until all the cards have been placed into one new pile. In this case, the first card will still be first, and the last card will still be last after the shuffle has been performed. For example, if your deck has 8 cards, labeled $\{0, 1, 2, 3, 4, 5, 6, 7\}$, after one out shuffle the deck will be $\{0, 4, 1, 5, 2, 6, 3, 7\}$

An *In Shuffle* is very similar to an out shuffle, but you select the first card from the second pile instead of the first. You then continue taking cards from one pile and then the second. If your deck, like before, consists of the cards $\{0,1,2,3,4,5,6,7\}$, after an in shuffle it will be $\{4,0,5,1,6,2,7,3\}$. It is easy to see that an out shuffle of $k+2$ cards is very similar to an in shuffle of k cards. While an out shuffle fixes the two outermost cards, the other k undergo an in shuffle.

In light of the above, it is not surprising that many properties of in and out shuffles can be derived from one another. In particular, it can be proven that the order of an out shuffle is the order of 2 which is equal to 1(mod 2n-1), while the order of an in shuffle is the order of 2 which equals 1(mod 2n+1)[Dia83].

Clearly the key property of both in and out shuffles is their preservation of central symmetry. In other words, elements that are in symmetric locations around the center of the deck will remain in symmetric locations after a perfect shuffle has been performed. For example, the in shuffle described above sends the pair originally in positions 0,7 to positions 1,6; the pair 1,6 to positions 3,4 etc. Because of this property, the *Shuffle Group*, i.e. the group generated by both in and out shuffles, can be viewed as a subgroup

of B_n,[1] the group of nxn signed permutation matrices. This is isomorphic to the group of nxn matrices with exactly one non-zero element in each row and each column, that element being plus or minus 1.

Let $< I, O >$ represent the shuffle group on a deck of $2n$ cards. The group $< I, O >$ has been completely determined for all values of n. With two exceptions, ($n = 6$ and $n = 12$), the nature of the group falls into one of five categories. Three of the five are related to three different homomorphisms of B_n.

Note that $sgn(g)$ is its sign as a permutation of $2n$ cards. $\overline{sgn}(g)$ is its sign as a permutation of n centrally symmetric pairs. And g $\rightarrow sgn(g) * \overline{sgn}(g)$ is a third homomorphism to $\pm$ 1.

The five categories are [Dia83]:

1. If n $\equiv$ 0(mod 4), $n > 12$ and $n \neq 2^k$, then $< I, O >$ is the intersection of the kernels of sgn and $\overline{sgn}$ and $| < I, O > | = n!2^{n-2}$.

2. If n $\equiv$ 1(mod 4), and n $\geq$ 5, then $< I, O >$ is the kernel of $\overline{sgn}$ and $| < I, O > | = n!2^{n-1}$.

3. If n $\equiv$ 2(mod 4), $< I, O >$ is isomorphic to B_n, and $| < I, O > | = n!2^n$.

[1]Note that this notation is used in a later section for an unrelated topic. We apologize to the reader, but feel it is important to remain consistent with the notation in [Dia83].

4. If n $\equiv$ 3(mod 4), then $| < I, O > |$ is isomorphic to the kernel of $sgn(g) * \overline{sgn}(g)$ and $| < I, O > | = n!2^{n-1}$.

5. If 2n=2^k, $< I, O >$ is isomorphic to the semi-direct product of Z_2^k by Z_k where Z_k acts as a cyclic shift, and $| < I, O > | = k * 2^k$

This information is summarized in the table below.

n	Isomorphic to	Size of Group
0(mod 4) ($n > 12$ and $2n \neq 2^k$)	intersection of the kernels of sgn and $\overline{sgn}$	$n!2^{n-2}$
1(mod 4) (n $\geq$ 5)	the kernel of $\overline{sgn}$	$n!2^{n-1}$
2(mod 4) ($n \neq 6$)	B_n	$n!2^n$
3(mod 4)	kernel of $sgn(g) * \overline{sgn}(g)$	$n!2^{n-1}$.
2n=2^k	the semi-direct product of Z_2^k by Z_k	$k * 2^k$ (=$2n * log2n$)

Figure 1

In the first four cases, $| < I, O > |$ is $O(n!2^n)$. In the fifth case, where $2n = 2^k$, $| < I, O > |$ has a significantly lower order of magnitude: $nlogn$ rather than $n!2^n$

In addition, determining whether a particular permutation on $2n$ cards belongs to the appropriate shuffle group $< I, O >$ is relatively easy in the

first four cases, but significantly harder when $2n = 2^k$.

If $2n \neq 2^k$, one only has to determine if the permutation preserves central symmetry (that is the ONLY requirement in case (1)), and whether it obeys one or two requirements of parity (cases 2-4).

If $2n = 2^k$, the requirements are much more complex. However, as indicated in the theorem, there is a complete characterization of the group $< I, O >$ in this case as well. Let $(x_1, ..., x_k)$ be the binary representation of any of the 2^k cards numbered 0 to $2^k - 1$. Then $O : (x_1, x_2, ..., x_k) \rightarrow (x_2, x_3, ..., x_k, x_1)$ and $I : (x_1, ..., x_k) \rightarrow (x_2, x_3, ..., x_k, x_1*)$ with $x_1* = 1 - x_1$

The permutations $O^j, 0 \leq j < k$ form a cycle group $< O >$ [Dia83]. Moreover, the permutations $B_j = O^{j-1}IO^{-j}$ send $(x_1, ..., x_j, ..., x_k)$ into $(x_1, ..., x_j*, ..., x_k)$. Finally, each element of the shuffle group has a unique representation of the form $O^j \prod_{i \epsilon S} B_i$ where S is some subset of $\{1, .., k\}$.

4.3 The Shuffle Group on Eight Cards

We will now look at some details for the group of order $N = 8 = 2^3$. Label the deck $\{0, 1, 2, 3, 4, 5, 6, 7\}$.

The order of the in shuffle is 6, because $2n + 1 = 9$ and $2^6 \equiv 1 \pmod 9$.

The sequence of in shuffles is as follows:

$$I : \{4, 0, 5, 1, 6, 2, 7, 3\}$$

$$I^2 : \{6, 4, 2, 0, 7, 5, 3, 1\}$$

$$I^3 : \{7, 6, 5, 4, 3, 2, 1, 0\}$$

$$I^4 : \{3, 7, 2, 6, 1, 5, 0, 4\}$$

$$I^5 : \{1, 3, 5, 7, 0, 2, 4, 6\}$$

$$I^6 : \{0, 1, 2, 3, 4, 5, 6, 7\} = ID$$

Notice that the deck reverses after 3 in shuffles. As we will see later, when $N = 2n = 2^k$, I^k always reverses the deck.

The order of the out shuffle is 3, because $2n - 1 = 7$, and $2^3 \equiv 1 \pmod 7$

The sequence of out shuffles is as follows:

$$O : \{0, 4, 1, 5, 2, 6, 3, 7\}$$

$$O^2 : \{0, 2, 4, 6, 1, 3, 5, 7\}$$

$$O^3 : \{0, 1, 2, 3, 4, 5, 6, 7\} = ID$$

Thus, in the group $< I, O >$ of shuffles generated by sequences of in and out shuffles, we have the two key relations: $I^6 = O^3 = ID$.

We will list all the elements in their canonical form when we deal with the conjugacy problem in a later section.

Chapter 5

The Word Problem

Generally speaking, the Word Problem is immediately solvable for the shuffle group. Given any expression in the generators I and O, one can simply determine the resulting permutation. Two expressions represent the same element if and only if they denote the same permutation.

In theory, at least, this observation applies to any finite group. As is well known, any such group is isomorphic to a subgroup of the permutation group on a finite number of objects. On the other hand, if the group is not originally defined as a permutation group, this idea may have no practical significance, since there would be no way to tell whether two strings define the same group element. In particular, if the group is defined by a set of generators and relations, the question would be whether the given relations could be

used to reduce any element of the group to a certain unique canonical form.

As we will see in the next section, the solution to the conjugacy problem is uniquely difficult for the Shuffle Group when $2n = 2^k$. Of course, since this group is finite, a solution does exist, but we will attempt to find a more efficient solution. As a prerequisite for the conjugacy problem, we will need an alternative solution to the word problem. For this reason, and because of the efficiency of the method, we will present this alternative solution to the word problem for $2n = 2^k$, using the group relations rather than the underlying permutations.

5.1 Solving the Word Problem for $2n = 2^k$ Using Group Relations

Using previously defined symbols, it is easy to see that $I = OB_k^{\cdot}$. Thus it is easy to convert any string of I's and O's into its canonical representation of the form $O^j \Pi_{i \epsilon S} B_i$, where $0 \leq j < k$ and $S \subset \{1, 2, ..., k\}$.

To establish the necessary identities, we introduce the following notation:

1. $B_S = \Pi_{i \epsilon S} B_i$

2. For $l \geq$ j, $B_{l,j}$ is equal to B_W where W is the set of j consecutive

integers whose largest value is l

3. $B_{k,k} = B_k B_{k-1} ... B_1$ will be denoted Z. It is the "reverser" in that $Z = (x_1, ..., x_k) = (x_1*, ..., x_k*)$. That is, Z maps any integer x into its centrally symmetric counterpart $2n - 1 - x = 2^k - 1 - x$.

4. $S \pm j$ will denote a translation of the integers in S, mod k, by $\pm j$

   i.e. $S \pm j = \{s_i \pm j \mid s_i \epsilon S\}$

5. $S + T = (S \bigcup T) - (S \bigcap T)$

   i.e. $S + T$ denotes the set of elements in S or in T but *not* in both.

The identities needed to reduce an element of $< I, O >$ to its canonical form are:

(I1) $B_i B_i$ = identity, for all i

(I2) $B_s O^j = O^j B_{s-j}$

This follows from the fact that $O^{-j} B_i O^j = B_{i-j}$ for any positive integer i, so that $O^{-j} B_s O^j = B_{s-j}$ for all subsets s. Multiplying the above expressions by O^j gives the identity.

(I3) For $1 \leq j < k$

$I^j = (OB_k)^j = O^j B_{k,j}$ (as defined above).

This can be seen directly or it can be proven by induction as follows:

The case $j = 1$ is the identity $I = OB_k$.

Assuming the identity for $j - 1$,

$I^j = II^{j-1} = OB_kO^{j-1}B_{k,j-1}$

$= O^jB_{k-j+1}B_{k,j-1} = O^jB_{k,j}$

Note that as a simple corollary of (I3), $I^k = B_{k,k}$ which is the reverser discussed above.

To simplify any string of the form

$$I^{f_n}O^{e_n}...I^{f_2}O^{e_2}I^{f_1}O^{e_1},$$

we use the following algorithm:

1. Reduce all integers e_n mod k, since O^k = identity

2. Reduce all integers f_n mod 2k, since I^{2k} = identity

3. If the reduced $f_n = j + k, 0 \leq j < k$ then

   $I^{f_n} = I^kI^j = ZO^jB_{k,j}$.

   Otherwise

   $I^{f_n} = I^j = O^jB_{k,j}$

4. Bring all factors of O to the left, making repeated use of identity (I2)

5. Simplify the products B_SB_T as B_{S+T}, since

   B_iB_i = identity for all integers i.

Example:

Suppose $k = 4$. Then $I^9O^7I^{14}O^5 = IO^3I^6O$

$= (OB_4)O^3ZI^2O$

$= (OB_4)O^3Z(O^2B_4B_3)O$

$= OB_4O^3ZO^3B_3B_2$

$= OB_4O^6ZB_3B_2$

$= OB_4O^2ZB_3B_2$

$= O^3B_2ZB_3B_2$

$= O^3ZB_3$

$= O^3B_1B_2B_4$

Notice that the power of O in the canonical form, namely 3, is the same $(mod 4)$ as the original sum, 35, of the powers of O and I.

In general, then, $I^{f_n}O^{e_n}...I^{f_2}O^{e_2}I^{f_1}O^{e_1}$

can be simplified by rewriting each power of I as $Z * O^jB_{k,j}$ or $O^jB_{k,j}$, and then shifting all powers of O to the left.

The ultimate form: O^jB_s will have $j \equiv (\Sigma_i(e_i + f_i))\ mod k$

We have now shown that any string of I's and O's can be reduced to its canonical form, thereby solving the word problem. Note, additionally, that the number of steps is a linear function of the number of changes from I to

O in the string.

5.2 Analysis of the Complexity of the Word Problem Algorithms

Even for finite groups, there is no obvious direct approach for solving the word problem (for that matter, it is not even easy to tell if a group defined by generators and relations is a finite group). Thus it is extremely helpful that the generators of the shuffle group are permutations. This allows us to solve the word problem in $< I, O >$ by checking for equality of the permutations associated with any two "words" in the generators.

The number of steps used in determining the permutation form is a linear function of the length l of any word. The number of steps involved in each successive application of I or O, and the number of steps in testing the equality of two permutations, are linear functions of n, where $2n$ is the number of cards in the underlying deck.

Since $| < I, O > |$ is finite, it makes sense to compare the efficiency of the algorithm to the order of the group. If $2n \neq 2^k$, then the above solution to the word problem, which is $\bigcirc(n)$, is of much smaller order of magnitude

than $| < I, O > |$, which is $\bigcirc(2^n * n!)$. If $2n = 2^k$, $| < I, O > |$ is only $k * 2^k$ or $\bigcirc(nlogn)$. In this case, the alternative method presented above, reducing each element of $< I, O >$ to its canonical form $O^j B_S$, is more efficient. The number of steps involved is again a linear function of the word length. In this algorithm, however, the only additional step required to solve the word problem is testing the equality of two subsets S, T of $\mathbb{Z}_k$. Thus the number of steps required is at most $\bigcirc(k)$. Since $2n = 2^k$, this is $\bigcirc(logn)$, while $| < I, O > | = k * 2^k = \bigcirc(nlogn)$.

Chapter 6

The Conjugacy Problem

6.1 The Case $2n \neq 2^k$

We will present two different algorithms for solving the conjugacy problem. The first method can be applied to four of the five different types of shuffle groups described previously. That is, as long as $2n \neq 2^k$ (and n is neither of the exceptional values, 6 or 12), we will use the underlying permutation form to determine if any two elements of $< I, O >$ are conjugate.

Note that two permutations of $2n$ objects are conjugate *in the Symmetric Group* S_{2n} if and only if they have the same cycle structure [Rot65]. In fact, all conjugates of a permutation f are found by permuting the elements $a_1, ..., a_{2n}$ while maintaining the same cycle structure. If two elements *do*

have the same cycle structure, say

$$f = (a_1, ..., a_{k_1})(a_{1+k_1}, .., a_{k_1+k_2})...(a_{k_{x-1}} + a_{k_x}, .., a_{k_x+k_{x+1}})$$

and

$$g = (b_1, ..., b_{k_1})(b_{1+k_1}, .., b_{k_1+k_2})...(b_{k_{x-1}} + b_{k_x}, .., b_{k_x+k_{x+1}})$$

then $f = hgh^{-1}$ if h is any permutation sending each element a_j to its corresponding element b_j. Generally, there are a great number of possibilities for the conjugator h since there are many equivalent ways to present the cycle structure of f, while maintaining its correspondence with the cycle structure of g. In particular, there are two changes to elements of f that are permissible:

1. We will label the cyclic shift of $(a_1\ a_2\ ...\ a_l)$ into $(a_2\ ...\ a_l\ a_1)$ a *rotation* of the elements. The elements in a cycle of length k can then be presented in $k-1$ additional forms corresponding to 1 through $k-1$ *rotations* of the elements.

2. The elements of any two cycles of the same length can be *switched.* This will be referred to as a *cycle switch.*

Analyzing their cycle structures can also be used to determine conjugacy of any 2 elements in a subgroup of S_{2n}, and particularly in $< I, O >$. As

in S_{2n}, the equivalence of the cycle structures is required. However, unlike the situation in S_{2n}, it must also be determined whether *at least one* of the possible permutations h, sending the elements of f to their counterparts in g, belongs to the subgroup. That is, in our case, we must check if h belongs to $< I, O >$.

To that end, we first examine some special properties of the cycle structure of any permutation in $< I, O >$.

6.1.1 The Cycle Structure of $< I, O >$

Definition 1. If $a+b = 2n-1$, a and b will be called *complementary numbers* and we will write $b = a'$. The set $\{a, b\}$ will be called a (complementary) couple.

Definition 2. If the set $\{a_1, ..., a_k\}$ contains no couples, the cycle $(a_1\ a_2\ ...\ a_k)$ is a *singles cycle.* If $k = 2j$ and if $\{a_1, ..., a_k\}$ consists of j couples, $(a_1\ ...\ a_k)$ will be called a *couples cycle.*

Lemma 1. *If $f \in < I, O >$, every cycle of f is either a singles cycle or a couples cycle of the form $(a_1\ ...a_j\ a_{j+1}\ ...\ a_{2j})$ with $a_{j+i} = a_i'$ for $i = 1, 2, ..., j$.*

Proof: Suppose a cycle of f contains a couple. By rotating the cycle, we may assume that the couple consists of a_1 and a_{j+1}. By the central symmetry

of f, since $a_{j+2} = f(a_{j+1})$ and $a_2 = f(a_1)$, $a_{j+2} = a'_2$. Similarly, $a_{j+3} = a'_3$, etc. and $f(a_{2j}) = a'_{j+1} = a_1$, completing the cycle.

Lemma 2. *Every singles cycle of f has a corresponding singles cycle consisting of the complementary numbers.*

Proof: By central symmetry, the cycle $(a_1\ a_2\ ...\ a_k)$ implies the existence of the complementary cycle $(a'_1\ a'_2\ ...\ a'_k)$.

6.1.2 Rotations and Switches

Suppose we are seeking to determine whether f and g are conjugate. As already mentioned, their cycle structures must be identical. Suppose, moreover, that h maps the elements a_j in the cycle structure of f into the corresponding elements b_j of g. Then, in order for h to satisfy the requirement of central symmetry, it is necessary and sufficient that the couples cycles of f are mapped into the couples cycles of g, and that complements of singles cycles in f are mapped into the complements of the corresponding singles cycles of g.

In the case where $n \equiv 2(mod4)$, this is the *only* requirement necessary to ensure $h_1\ \epsilon < I, O >$. However, for other values of n, there are one or two additional requirements, which may or may not be satisfied.

Example with n=3:

f = (0 1) **(2 3)** (5 4)

$\downarrow h_1$

g = (0 2) **(1 4)** (5 3)

h_1 = (1 2) (3 4)

Note that h_1 preserves central symmetry but is not a member of $< I, O >$ since $sgn(h_1) * \overline{sgn}(h_1) \neq 1$.

In the above example, however, if we change the order of the elements of f by rotating the cycle (2 3), the resulting mapping h_2 now has the cycle structure h_2 = (1 2 4 3) and $h_2 \in < I, O >$ since $sgn(h_2) * \overline{sgn}(h_2)$ = $(-1)(-1) = +1$. Thus, in fact, f and g are conjugate.

To generalize the above example, note that if a rotation or switch (R or S) is first applied to the order of the elements in the cycle structure of f, and the elements of f in the new order are then mapped by h_2 to the elements in the corresponding cycle structure of g, then (R or S)$h_2 = h_1$. Hence $sgn(h_2)$ will equal $sgn(h_1)$ and $\overline{sgn}(h_2) = \overline{sgn}(h_1)$ if and only if the corresponding sgn or $\overline{sgn}$ of R or S is even. Thus, to determine if there is some h in the shuffle group with $f = hgh^{-1}$, we must first analyze the parity of every permissible rotation or switch which can be applied to cycles of f.

The following eight lemmas enumerate all possibilities. In many cases the proofs are immediate and are therefore omitted.

In all cases we will let l denote the length of a cycle.

Lemma 3. *If l is odd, and R denotes the rotation of two complementary l-cycles, then $sgn(R)$ and $\overline{sgn}(R)$ are both $=+1$.*

Proof: R is given by one *l-cycle* in S_n, and since l is odd, the cycle is an even permutation. R equals two *l-cycles* in S_{2n}.

Example with $n = 4$:

(0 1 3)(7 6 4)

$h \downarrow$

(1 3 0)(6 4 7)

has cycle structure (0 1 3) viewed as an element of S_n and cycle structure (0 1 3)(7 6 4) in S_{2n}.

Lemma 4. *If l is odd, and S denotes a switch of two complementary l-cycles, then $sgn(S) = -1$ while $\overline{sgn}(S) = +1$.*

Proof: Note that S represents l transpositions in S_{2n} and it is the identity in S_n.

Lemma 5. *If l is odd, and S denotes a switch of two* non-*complementary*

l-cycles along with a switch of their complementary l-cycles, then $sgn(S)=+1$ *and* $\overline{sgn}(S)=-1$.

Proof: S represents $2l$ transpositions in S_{2n} and l transpositions in S_n.

Lemma 6. *If l is even, and R rotates a singles cycle of length l along with its complement, then* $sgn(R)=+1$ *and* $\overline{sgn}(R)=-1$.

Proof: R is a product of two *l-cycles* in S_{2n} and is a single *l-cycle* in S_n.

Lemma 7. *If l is even, and S switches a singles cycle with its complement, then* $sgn(S) = \overline{sgn}(S) = +1$.

Lemma 8. *If l is even, and S switches two* non-*complementary singles cycles with each other, along with their complementary cycles, then* $sgn(S)=\overline{sgn}(S)=+1$.

Lemma 9. *If l is even, and R rotates a couples cycle of length l,*

then $sgn(R)=-1$;

$\overline{sgn}(R) = -1$ *if* $l \equiv 0(mod4)$ *and*

$\overline{sgn}(R) = +1$ *if* $l \equiv 2(mod4)$.

Proof: Note that in this case R represents a cycle of length $l/2$ in S_n.

Lemma 10. *If l is even, and S switches two couples cycles of length l with each other, then* $sgn(S) = +1$;

$\overline{sgn}(S) = -1$ *if* $l \equiv 2(mod4)$ *and*

$\overline{sgn}(S) = +1$ *if* $l \equiv 0(mod4)$.

Proof: Note that in S_n, S consists of $l/2$ transpositions.

In the following chart, we will list all types of permutations for which we do not have both sgn and $\overline{sgn}$ equal to $+1$.

	sgn(S or R)	$\overline{sgn}$(S or R)
l odd, S switches two comp. cycles	-1	+1
l odd, S switches two non-comp. l-cycles and their comp.s	+1	-1
l even, R rotates two comp. singles of length l	+1	-1
l even, R rotates one couples cycle of length l	-1	-1 if $l \equiv 0(\text{mod } 4)$ +1 if $l \equiv 2(\text{mod } 4)$
l even, S switches two couples cycles of length l	+1	+1 if $l \equiv 0(\text{mod } 4)$ -1 if $l \equiv 2(\text{mod } 4)$

Figure 2

6.1.3 The General Theorem

We are now ready to state our general theorem on the conjugacy of elements in $< I, O >$. Note that a necessary condition for conjugacy is that *not only* should the two permutations have the same cycle structure, but that they should have the same number of singles cycles and the same number of couples cycles of each length l.

In that case, we will say that the two cycle structures are *compatible*. As an example, consider the "reverser" which changes the order of the cards from $(0, 1, 2, ..., 2n-1)$ to $(2n-1, 2n-2, ..., 1)$. Its cycle structure consists

of the n couples transpositions:

(0 $2n$-1)(1 $2n$-2)...

The only possible compatible cycle structure would be a simple rearrangement of the same n transpositions. Thus the reverser has no conjugates.

Our general result, then, is as follows:

Theorem 1. *Suppose* $2n \neq 2^k$, $n \neq 6$ *or* 12. *If the cycle structures of* f *and* g *are not compatible, they are not conjugate. If their cycle structures* are *compatible, then:*

(a): If $n \equiv 2(mod\ 4)$ *or* $n \equiv 3(mod\ 4)$, *then* f *and* g *are conjugate.*

(b): If $n \equiv 1(mod\ 4)$, *then* f *and* g *are conjugate,* unless *the cycle structures of both* f *and* g *consist of one odd* $l-$*cycle and its complement. In that case, consider* any h *which maps the elements in a cycle structure of* f *onto the elements in the corresponding elements of* g. *Then* f *and* g *are conjugate if and only if* $h \epsilon < I, O >$. *That is, if* $h \epsilon < I, O >$, $f = hgh^{-1}$. *If* h *is not a member of* $< I, O >$, *there is no other possible conjugator of* f *and* g.

(c): If $n \equiv 0(mod\ 4)$, f *and* g *are conjugate unless the cycle structure of* f *and* g *consists of either one pair of complementary cycles of odd length, or singles cycles of even length only. The details in this case are enumerated in the proof below.*

Proof:

(a): If $n \equiv 2(\text{mod } 4)$, any h which maps elements of f to corresponding elements in g, with complementary singles cycles mapped into complementary singles cycles, will have central symmetry. Hence $h \in < I, O >$ since in this case $< I, O >$ is equal to the full group of centrally symmetric permutations on $2n$ objects and f and g are conjugate.[Dia83].

If n $\equiv$ 3(mod 4), $h \in < I, O >$ if and only if $sgn(h) * \overline{sgn}(h)$=1. Suppose then that such a mapping h_1 has $sgn(h_1) * \overline{sgn}(h_1)$=-1. We claim that it is possible to first apply an appropriate R or S to the elements in the cycle of f so that the resulting mapping h_2 will have $sgn(h_2) * \overline{sgn}(h_2) = +1$:

- If the cycle structure of f has at least one odd cycle, apply a switch.
- If the cycle structure of f contains even one even singles cycle, apply a rotation.
- Finally, suppose the only cycles in f are couples cycles. This cycle structure, however, is not possible when $n \equiv 3(\text{mod } 4)$. This follows from the following lemma:

Lemma 11. *If n is odd and $g \in < I, O >$ on a deck of $2n$ cards, and if the cycle structure of g consists of only couples cycles, then $sgn(g) * \overline{sgn}(g)$=-1.*

(And hence n cannot be congruent to 3(mod4)).

Proof:

Suppose the k distinct cycles have $2n_1$, $2n_2$, ... , $2n_k$ elements. Then, since each cycle is composed of couples in S_n, g can be viewed as the result of $(n_1-1)+(n_2-1)+...+(n_k-1)=(n-k)$ transpositions. In S_{2n}, it can be viewed as $(2n_1-1)+(2n_2-1)+...+(2n_k-1)=(2n-k)$ transpositions. Since n is odd, the parity for the number of transpositions in the two different cases cannot be the same.

(b): If $n \equiv 1(\text{mod } 4)$, $h \in < I, O >$ if and only if $\overline{sgn}(h)$=+1.

A quick look at the last column of our chart above shows that if $\overline{sgn}(h)$=-1, by preceding h with the appropriate R or S, we can always change it so that we will find an appropriate $h \in < I, O >$. The only exception is when f consists of *only* one odd cycle and its complement. In that case, $sgn(h)$ is unaffected by the only possible switch available, and the two elements f and g will be conjugate if and only if any particular h mapping the elements of f to the elements of g belongs to the shuffle group.

Example: If $n = 9$, and if

f = (0 1 2 3 4 5 6 7 8) (17 16 15 14 13 12 11 10 9)

g = (1 2 3 0 4 5 6 7 8) (16 15 14 17 13 12 11 10 9)

then the mapping h from the elements of f to the corresponding elements of g has cycle structure h = (0 1 2 3), and $\overline{sgn}(h)$=-1. In this case, no rotation nor cycle switch of f will produce a resulting h that belongs to the shuffle group. Hence f and g are not conjugate in the shuffle group.

(c): If $n \equiv 0(\text{mod } 4)$, $h \in < I, O >$ if and only if $sgn(h) = \overline{sgn}(h)$=+1. Suppose then that h is a mapping between f and g and if h is not an element of $< I, O >$. A quick look at the chart shows the following breakdown:

- If $sgn(h)$ = -1, and the cycle structures of f and g consist *entirely* of singles cycles of even length, then f and g are not conjugate.
- If $\overline{sgn}(h)$ = -1, and the cycle structures of f and g consist *entirely* of one pair of complementary singles cycles of odd length, then f and g are not conjugate.
- If both $sgn(h)$ and $\overline{sgn}(h)$ are = -1, then either of the cycle structures described above would render f and g non-conjugate.

In the three cases above, however, for a particular f and g, one could determine conjugacy by simply aligning the elements of f and g in compatible cycle structures and considering the mapping h between the elements of f

and g. f and g are conjugate if and only if that particular h belongs to $< I, O >$.

6.1.4 Generalizing the Method

The method described above would seem to be applicable for solving the conjugacy problem in any subgroup of S_{2n}. A necessary condition for conjugacy would be that f and g have the same cycle structures. If they do, one would need only determine if *any* of the possible mappings between the corresponding elements and their cycle structures belongs to the subgroup under consideration.

The number of such possible mappings would often be of much smaller order of magnitude than the order of the underlying subgroup.

For general finite groups, the method would seem to be impractical. To associate the elements of the group with permutations, one would need to know the group table. Given the group table, however, one could simply apply Golomb's Algorithm to determine conjugacy.

In the shuffle group with $2n = 2^k$, the method seems likewise impractical. For one thing, there is no immediate way to determine from its cycle structure if an element belongs to the shuffle group. Moreover, the number of possible

mappings between two equivalent cycle structures f and g would almost always exceed the order of the shuffle group which, as noted before, is only of the order of $nlogn$.

For example, if the cycle structures of f and g had as few as two non-complementary cycles of length $O(n)$, then there would be at least $O(n^2)$ possible mappings to consider. Moreover, if there were $O(n)$ cycles of the same length, then in light of the possible cycle switches, the number of mappings to be considered would be of exponential order!

The next section indicates an alternative approach to the conjugacy problem when $2n = 2^k$.

6.2 Solving the Conjugacy Problem When $2n = 2^k$

The basic strategy will be to reduce any two elements of the shuffle group $< I, O >$ to their canonical representations $O^j B_S$ and $O^m B_W$. It will then follow that the two elements are conjugate if and only if $m = j$ and the two sets W and S are related in a manner to be described.

Since every element of the shuffle group $< I, O >$ (when $2n = 2^k$) is of

the form O^pB_T, the most general conjugate of O^jB_S is given by

$(O^pB_T)(O^jB_S)(O^pB_T)^{-1}$

$=O^pB_TO^jB_SB_TO^{-p}$

$=O^pB_TO^jO^{-p}B_{S+p}B_{T+p}$

$=O^jB_{T+p-j}B_{S+p}B_{T+p}$

$=O^jB_{V+p}$ where $V = S + T + (T - j)$.

Thus we have

Theorem 2. *O^mB_w will be conjugate to O^jB_s if and only if $m = j$ and $W = V + p$, where $V = S + T + (T - j)$ for some subset T of $\mathbb{Z}_k$*

Several immediate corollaries follow.

Corollary 1. *Any element of the form B_S (i.e. any element with j=0 in its canonical form) belongs to a conjugacy class consisting of $\{B_W\}$ where $W = S + p$ for any integer p.*

This follows from the fact that $S + T + T = S$. See the example below for $k = 3$.

Corollary 2. *Suppose an element O^jB_S forms its own conjugacy class. Then clearly $j = 0$ and moreover $S+p$ must equal S, for all p. This is only possible, however, if $S = \emptyset$ or $S = \{1, ..., k\}$.*

Thus the only single element conjugacy classes are the identity and $\{Z\}$ (the "reverser"). That is, the center of the group consists of the identity and Z.

As an illustrative example, if $k = 3$, the conjugacy classes are:

$\{identity\}$

$\{B_1, B_2, B_3\}$

$\{B_1B_2, B_2B_3, B_1B_3\}$

$\{Z\} = \{B_1B_2B_3\}$

$\{O, OB_1B_2, OB_2B_3, OB_1B_3\}$

$\{OB_1, OB_2, OB_3, OB_1B_2B_3\}$

$\{O^2, O^2B_1B_2, O^2B_2B_3, O^2B_1B_3\}$

$\{O^2B_1, O^2B_2, O^2B_3, O^2B_1B_2B_3\}$

Note that, in the above case, the order of $< I, O >$ is $k * 2^k = 24$ and, of course, the order of every conjugacy class is a divisor of the order of the group.

Corollary 3. *There will always be a conjugacy class of order k. If k is an odd prime, the number of such conjugacy classes must be even.*

Proof: The existence of at least one such conjugacy class is a direct

result of our earlier results, i.e. $\{B_1, B_2, ..., B_k\}$ is a conjugacy class. Since the order of every conjugacy class is a divisor of the order, $k * 2^k$, of the group, every conjugacy class must have order 1, order k, or some even order. Since, however, there are exactly 2 conjugacy classes of order 1, and since the sum total of all the elements equals the even number $k * 2^k$, it follows that there must be an even number of the conjugacy classes of the prime odd order k.

Note: Suppose we say S and W are *j-conjugate* subsets if $W = V + p$, where $V = S + T + T - j$, for some subset T of $\mathbb{Z}_k$ and some integer p. If $j = 0$, W is simply a translation of S. In other cases, the situation is slightly more complicated. However, since T and $T - j$ have the same number of elements, it follows that the orders of conjugate sets S and W must always have the same parity - a fact illustrated by the above example.

6.2.1 A Lower Bound for The Number of Conjugacy Classes

It is possible to determine a lower bound for the number of conjugacy classes in the shuffle group with deck size $2n = 2^k$. This is of some interest since we are trying to disguise an element by sending one of its conjugates in its

place. The ideal environment would be a group with very few conjugacy classes, each with many elements. In a group with deck size 2^k, as noted before, the elements have the canonical form $O^j B_S$, where $j \in \{0, 1, ..., k-1\}$ and S is a subset of $\mathbb{Z}_k$.

We have determined that elements, $O^j B_S$ and $O^m B_W$ are conjugate if and only if $m = j$, and the sets S and W are "*j-conjugate*". For $j = 0$, as noted before, the sets S and W must be simple shifts of each other. Thus, there is at least one separate conjugacy class corresponding to each of the $k+1$ possible orders of the set S. Moreover, while $k+1$ is the exact number of conjugacy classes if $k = 1, 2$ or 3, a much greater lower bound can be given for large k. Determining the number of subsets of $\mathbb{Z}_k$ which are not translates of each other involves Polya enumeration and is somewhat complicated. It is easy to see, however, that this number exceeds $2^k/k$. This follows from the observation that among the 2^k subsets of $\mathbb{Z}_k$, no distinct conjugacy class corresponds to more than k (the maximum number of distinct translates of a set).

For each $j > 0$, there are at least two different conjugacy classes, corresponding to the different possibilities for the parity of S. (As we will see below, this is the exact number of conjugacy classes if j and k are relatively

prime).

Adding the above results yields the following:

Proposition 1. *If* $2n = 2^k$, *the number of distinct conjugacy classes is at least as large as* $2(k-1) + Max(k+1, 2^k/k)$.

Returning to the question of determining the possible conjugacy of two elements, we seek to characterize the relationship of $j - conjugacy$ between sets S and W:

$W = V + p$, $V = S + T + (T - j)$.

Note that $j - conjugacy$ of sets is clearly an equivalence relation.

1. $j = 0$

   Two sets are 0-*conjugate*, as already noted, if they are translates of each other.

2. $j = 1$

   Two sets are 1-*conjugate*, as we will demonstrate below, if and only if they have the same parity.

   Proof:

   a) Any set with an even number of elements is 1-*conjugate* to the empty set. If the set S has elements $\{a_1, a_2, ..., a_{2N-1}, a_{2N}\}$, then $S = \emptyset + T +$

$(T-1)$, where $T = \{1+a_1, ...a_2, 1+a_3, ..., a_4, ..., 1+a_{2N-1}, ..., a_{2N}\}$.

Example:

The set $\{1,4,7,11\} = T + (T-1)$ with T=$\{2,3,4,8,9,10,11\}$.

b) Any set with an odd number of elements is conjugate to the set $\{1\}$.

If $S = \{a_0, a_1, a_2, ..., a_{2N}\}$, then S=$\{1\}$ + $T + (T-1)$, where

T=$\{2, .., a_0, 1+a_1, .., a_2, ..., 1+a_{2N-1}, ..., a_{2N}\}$.

3. $j = 2$

In order to determine $2-conjugacy$, we must divide S into S_1, consisting of the odd elements of S, and S_0, consisting of the even elements of S.

a) If the order of S is odd, then either $|S_0|$ or $|S_1|$ is odd, and the other one is even. To show that S is conjugate to the set $\{1\}$, assume w.l.o.g. that S_0 equals the set $\{a_1, a_2, ..., a_{2n}\}$ and S_1 equals $\{b_0, b_1, ..., b_{2n}\}$. Then a translation of $\{1\}$ by the number $b_0 - 1$ will give us $\{b_0\}$. Moreover, all the other elements of S can be added to b_0 by the addition of the appropriate T and $T-2$ (since $j = 2$).

Example:

$\{6, 50, 60, 7, 11\}$=$\{6\}$ +T+$(T-2)$ where $T = \{52, 54, 56, 58, 60, 9, 11\}$.

b) If the order of S is even, then, assuming $|S_0|$ and $|S_1|$ are both even, S is conjugate to $\emptyset$, by pairing elements as in the above argument.

If $|S_0|$ and $|S_1|$ are both odd, say

$S_0 = \{a_0, ..., a_{2N}\}$ and

$S_1 = \{b_0, ..., b_{2N}\}$

then S is clearly conjugate to $\{a_0, b_0\}$, which in turn is conjugate to $\{1, 2\}$. We now have to consider two cases.

If k is even, $\{1, 2\}$ represents a distinct conjugacy class which *does not* contain the empty set.

On the other hand, if k is odd, $\{1, 2\}$ is conjugate to the empty set, since $\emptyset = T + (T - 2)$, with $T = \{4, 6, 8, .., k - 1, 1\}$ since $k + 1 \equiv 1 (\text{mod } k)$.

The following box summarizes the cases for $j = 2$:

$\lvert S_0 \rvert$	$\lvert S_1 \rvert$	S is conjugate to
even	odd	$\{1\}$
odd	even	$\{1\}$
even	even	$\{0\}$
odd	odd	$\{0\}$ if k is odd
		$\{1,2\}$ if k is even

Figure 3

These results for $j = 3$ are summarized in the table below.

$\lvert S_0 \rvert$	$\lvert S_1 \rvert$	$\lvert S_2 \rvert$	S is conjugate to
even	even	even	$\{0\}$
even	even	odd	$\{1\}$
odd	even	even	$\{1\}$
even	odd	even	$\{1\}$
odd	odd	even	$\{0\}$ if $k \neq 0 (\text{mod } 3)$
			$\{1,2\}$ otherwise
odd	even	odd	$\{0\}$ if $k \neq 0 (\text{mod } 3)$
			$\{1,2\}$ otherwise
even	odd	odd	$\{0\}$ if $k \neq 0 (\text{mod } 3)$
			$\{1,2\}$ otherwise
odd	odd	odd	$\{1\}$ if $k \neq 0 (\text{mod } 3)$
			$\{1,2,3\}$ otherwise

Figure 4

Note that S_0 now indicates the elements in S that are congruent to 0(mod3).The elements of S_1 are congruent to 1(mod 3) and S_2 consists of the elements which are congruent to 2(mod 3).

The above examples are all special cases of a general theorem which we will prove based on the following proposition. We noted previously that $T + (T - j)$ will always have an even number of elements. The proposition offers a more complete picture.

Proposition 2. *Let T range over all subsets of $\mathbb{Z}_k$. Then the corresponding sets $U = T + (T - j)$ represent all sets with an even number of elements in each congruence class mod d, where $d = gcd(j, k)$.*

Proof:

For any $x \epsilon T$, the set $T + (T - j)$ contains both x and $x - j$. Since both j (and k) are multiples of d, x and $x - j$ belong to the same congruence class *mod* d. Thus $U = T + (T - j)$ contains an even number of elements in each class. Suppose moreover that x and y are any two elements with $x \equiv y(mod\ d)$. Then for some integer a, $y = x - ad$. However, since $d = gcd(j, k)$, there exist integers r and s such that $rj + sk = ad$. Thus $rj \equiv ad(mod\ k)$ and if

$\{x, x - j, ..., x - (r - 1)j\} \subset T$

the corresponding set of points in U will be $\{x, y\}$.

According to the proposition, for any set S in $\mathbb{Z}_k$, there exists a unique "reduced set" $V = S + T + (T - j)$ containing precisely those elements t, $1 \leq t \leq d$, for which S contains an odd number of elements in that congruence class *mod d*.

Example:

Suppose $k = 9$ and $j = 6$ so that $d = 3$.

If $S = \{1, 2, 4, 5, 8, 9\}$, we can rewrite

$S = \{1, 4; 2, 5, 8; 9\}$, grouping the elements by their congruence class mod 3. Then with $T = \{1, 5, 9\}$,

$U = T + (T - j) = \{1, 4; 5, 8; 3, 9\}$ and $V = S + U = \{2, 3\}$.

Finally, to determine if two sets S and W are *j-conjugate*, note that each is conjugate to its "reduced set" (*mod d*), and we need only check if these are translates of each other. Combined with earlier results, this proves

6.2.2 The General Result

Theorem 3. *If $2n = 2^k$, two elements with canonical forms $O^j B_S$ and $O^m B_W$ are conjugate if and only if*

1. *m=j*

2. *the "reduced sets" of S and W (mod d) are translates of each other.*

Corollary 4. *If $2n = 2^k$ and k is prime, then for each $j > 0$, all elements of the form $O^j B_S$ fall into exactly two conjugacy classes.*

Proof:

Since k is prime, $gcd(j, k) = 1$ for all k and the two conjugacy classes correspond to the possible subsets of $\{1\}$.

6.3 Analysis of the Complexity of the Conjugacy Problem Algorithms

The conjugacy problem for finite groups can always be solved directly **if** one knows the group table. In that case, one can avoid the extra steps necessary in computing conjugates by using Golomb's algorithm to check for conjugacy. The number of steps required by this method, however, is proportional to the square of the order of the group.

The method outlined above for $2n \neq 2^k$ requires that we determine the cycle structure of the respective permutations. This involves $\bigcirc(n)$ operations. If $n \equiv 2 \pmod 4$ or if $n \equiv 3 \pmod 4$, conjugacy is then determined by checking the "compatibility" of the two permutations. That is, they must have identical cycle structures for each of the two types of cycles - singles

and couples. If $n \equiv 1 \pmod 4$ or $n \equiv 0 \pmod 4$, an additional check of $\overline{sgn}(h)$ and, possibly $sgn(h)$ is required for one permutation $h \in S_{2n}$. In all of these cases, the number of steps is $\bigcirc(n^2)$. As in the solution of the word problem, this is of much smaller order of magnitude than $| < I, O > |$ which is $\bigcirc(n! * 2^n)$.

If $2n = 2^k$, testing the possible conjugacy of two elements in canonical form involves partitioning two subsets of $\mathbb{Z}_k$ into separate congruence classes mod d, where d=$gcd(j, k)$. This process, and the subsequent check whether the two "reduced" sets are translations of each other, require $\bigcirc(k)$ steps. Since $2n = 2^k$, this is $\bigcirc(logn)$ and again is much smaller than $| < I, O > |$ which is $\bigcirc(k * 2^k)$ or $\bigcirc(nlogn)$.

Chapter 7

The Shuffle Group as a Platform for a Key-Exchange Protocol

One of the main features of the secret-key protocol described in section 3.4 is the fact that conjugation is a one-way function. That is, given elements a and g, one can easily compute $b = gag^{-1}$. On the other hand, knowing a and b does not determine g. In fact, solutions of the conjugacy equation $b = gag^{-1}$ form a coset of the centralizer of a, namely: $\{g_1 c | g_1 a g_1^{-1} = b$ and $cac^{-1} = a\}$.

Recall that in the secret key-exchange protocol, Alice has formed her

secret word X in the generators a_1, a_2, ..., a_n while Bob has formed his secret word Y in the generators b_1, b_2, ..., b_m. Over insecure channels, Alice transmits her elements a_1, ..., a_n to Bob, who replies with the conjugates of each a_i by Y. In like manner, Bob transmits b_1, b_2, ..., b_m and receives the X-conjugates of b_i in reply. Both Alice and Bob can then form their commutator $XYX^{-1}Y^{-1}$ as described in section 3.4. To maintain the privacy of the exchange, it is important that an eavesdropper not be able to solve the set of equations

$$Ya_iY^{-1} = A_i,\ Xb_jX^{-1} = B_j$$

for the unknowns X and Y.

In finite groups, one can check each element of the group for solutions to the equations above. For that reason, it is necessary that each of the elements a_i and b_j has a large centralizer. Thus, if the underlying group is the shuffle group, both Alice and Bob could choose elements of $< I, O >$ which have many smaller cycles rather than fewer long cycles. For example, an element consisting of one cycle would have only $2n$ elements in its centralizer.

Suppose then that an eavesdropper searches the group for elements u and v which satisfy

$$ub_ju^{-1} = B_j\ \forall\ j,\ va_iv^{-1} = A_i\ \forall\ i$$

It is still possible that the solutions u and v are not unique. In other words, it is possible that u and v are **not** X and Y respectively. Nevertheless, if either $\{a_i\}$ or $\{b_j\}$ form a set of generators of $< I, O >$, then

$$uvu^{-1}v^{-1} = XYX^{-1}Y^{-1}$$

This follows from the fact that

$$ub_ju^{-1} = B_j \; \forall \; j$$

implies $u = Xc_1$, where c_1 is the centralizer of **every** b_j.

Similarly, $v = Yc_2$, where c_2 commutes with every element a_i. Thus, c_1 commutes with every element in the group $< b_1, ..., b_m >$ and c_2 commutes with all elements of $< a_1, ..., a_n >$. Suppose one of these, say $< a_1, ..., a_n >$, equals the entire group G. Then c_2 is in $\mathbf{Z}(\mathbb{G})$ and

$$\begin{aligned} uvu^{-1}b^{-1} &= (Xc_1)(Yc_2)(c_1^{-1}X_1^{-1})(c_2^{-1}Y_2^{-1}) \\ &= Xc_1Yc_1^{-1}X^{-1}Y^{-1} \end{aligned}$$

since c_2 commutes with all elements of G.

$$= XYX^{-1}Y^{-1}$$

since c_1 commutes with $< b_1, ..., b_m >$, which includes Y.

Thus to maintain the difficulty of breaking the secret key-exchange protocol, when the underlying group is finite:

1. all elements a_i and b_j should have large centralizers

2. the intersection of the centralizers of $\{a_i\}$ as well as the intersection of the centralizers of $\{b_j\}$ should be non-trivial sets

3. neither $\{a_i\}$ nor $\{b_j\}$ should contain a set of generators of the underlying group.

Golomb's Algorithm - Code

```
//GOLOMB.CPP ************** find conjugates using Golomb's method

#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <conio.h>
#include "apmatrix.h"
#include "apstring.h"


const int SIZE = 6;
const char FILENAME \sqsubset \sqsupset="DIHEDRAL.DAT";
apmatrix<int> a4(SIZE,SIZE);
bool conjugate(int,int);
void displayTable();
void getElem();
void getFile();
void getOneElem();
void allConjugates(int);

void main()
{
    clrscr();
    getFile();
    displayTable();
    getElem();
    char next;
    for(;;) {
        cout << "Do you want to:" << endl;
        cout << "(1) Check another pair or" << endl;
        cout << "(2) Find all the conjugates of an element or" << endl;
        cout << "(3) Exit?" << endl;
        cin >> next;
        if ((next != '1') AND (next != '2')) break;
        displayTable();
        if (next == '1') getElem(); else getOneElem();
```

```
        }
}

//**********************

bool conjugate(int a, int b)
//check two elements and see if they're conjugate
{
    bool conj=false;
    for (int i=2;i<SIZE;i++)
    {
        for (int j=1;j<i;j++)
        {
            if (a4[i][j]==a)
            {
                if (a4[j][i]==b)
                {
                    cout << i << '*' << j << " = " << a ;
                    cout << j << '*' << i << " = " << b << endl;
                    conj=true;
                }
            }
        }
    }
    return conj;
}

//**********************

void allConjugates(int a)   //find all conjugates of an element
{
    int b;
    for (int i=0;i<SIZE;i++)
    {
        for (int j=0;j<i;j++)
        {
            if (a4[i][j]t==a)
            {
```

```
                b = a4[j][i];
                cout << i << '*' << j << " = " << a ;
                cout << j << '*' << i << " = " << b << endl;
            }
        }
    }
}

//**********************

void displayTable()         //displays the group table.
{
    cout << FILENAME << endl;
    for (int i=0;i<SIZE;i++)
    {
        for (int j=0;j<SIZE;j++)
            cout << setw(4) << (a4[i][j]);
        cout << endl;
    }
    cout << endl;
}

//**********************

void getElem()          //get two elements from user
{
    int w1,w2;
    cout << "What are the two elements that you are checking"
            << " for conjugacy?";
    cin >> w1 >> w2;
    if (w1==w2) {
        cout << "An element is always conjugate to itself";
        return;
    }
    if (conjugate(w1,w2))
        cout << "They are conjugate";
    else
        cout << "They are not conjugate";
```

```
    cout << endl << endl << endl;
}

//***********************

void getOneElem()   //get one element and find its conjugates
{
    int w1;
    cout << "What is the element whose conjugates you are looking for?";
    cin >> w1;
    allConjugates(w1);
    cout << endl << endl << endl;
}

//***********************

void getFile()      //open group table
{
    ifstream inpFile(FILENAME);
    if (!inpFile) cout << "Can't find the file";
    for (int i=0;i<SIZE;i++)
    {
        for (int j=0;j<SIZE;j++)
        {
            inpFile>>(a4[i][j]);
        }
    }
}
```

Determining the Cycle Structure, In and Out Shuffling

```
// SHUFFLE.H ********** header file

#ifndef SHUFFLE_H
#define SHUFFLE_H

#include "apstring.h"
#include "apmatrix.h"
#include "apvector.h"

int DECK_SIZE=8;

struct Deck{
    int element;
    bool checked;
};

class Shuffle{
private:
    int decksize;
    apvector<Deck> myDeck;
    apstring name;
    apmatrix<int> myCycles;
    ShowCycles();
public:
    InitDeck();
    ChangeDeckSize(int);
    ResizeDeck(int);
    Shuffle(int);
    Print();
    OutShuffle();
    InShuffle();
    InverseOutShuffle();
    InverseInShuffle();
    ShuffleString();
};
```

```
#endif


// SHUFFLE.CPP  ****************************

#include <fstream.h>
#include <iostream.h>
#include <iomanip.h>

#include "shuffle.h"

Shuffle::Shuffle(int size)
{
    decksize=size;
    myDeck.resize(decksize);
    myCycles.resize(decksize,decksize);
    for (int i=0;i<decksize;i++)
    {
        myDeck[i].element=i;
        myDeck[i].checked=false;
    }
}

Shuffle::InitDeck()
{
    myDeck.resize(decksize);
    myCycles.resize(decksize,decksize);
    for (int i=0;i<decksize;i++)
    {
        myDeck[i].element=i;
        myDeck[i].checked=false;
    }
    Print();
    SetCycles();
}

Shuffle::ChangeDeckSize(int newsize)
```

```
{
    decksize=newsize;
}

Shuffle::ResizeDeck(int newsize)
{
    myDeck.resize(newsize);
}

Shuffle::Print()
{
    for (int j=0;j<decksize;j++)
    cout << setw(4) << myDeck[j].element;
    cout << endl;
    ShowCycles();
}

Shuffle::OutShuffle()
{
    Shuffle Shuffled(decksize);
    for (int j=0;j<decksize/2;j++)
        Shuffled.myDeck[2*j].element=myDeck[j].element;
    for (int k=decksize/2;k<decksize;k++)
        Shuffled.myDeck[2*(k-(decksize/2))+1].element=myDeck[k].element;
    cout << setw(11) << "OUT: ";
    Shuffled.Print();
    *this=Shuffled;
}

Shuffle::InShuffle()
{
    Shuffle Shuffled(decksize);
    for (int j=0;j<decksize/2;j++)
        Shuffled.myDeck[2*j+1].element=myDeck[j].element;
    for (int k=decksize/2;k<decksize;k++)
        Shuffled.myDeck[2*(k-(decksize/2))].element=myDeck[k].element;
    cout << setw(11) << "IN: ";
    *this=Shuffled;
```

```
    Print();
}

Shuffle::InverseOutShuffle()
{
    Shuffle UnShuffled(decksize);
    for (int j=0;j<decksize/2;j++)
        UnShuffled.myDeck[j].element=myDeck[2*j].element;
    for (int k=decksize/2;k<decksize;k++)
        UnShuffled.myDeck[k].element=myDeck[k*2-decksize+1].element;
    cout << setw(11) << "INVERSE OUT: ";
    UnShuffled.Print();
    *this=UnShuffled;
}

Shuffle::InverseInShuffle()
{
    Shuffle UnShuffled(decksize);
    for (int j=0;j<decksize/2;j++)
        UnShuffled.myDeck[j].element=myDeck[j*2+1].element;
    for (int k=decksize/2;k<decksize;k++)
        UnShuffled.myDeck[k].element=myDeck[(k-decksize/2)*2].element;
    cout << setw(11) << "INVERSE IN: ";
    UnShuffled.Print();
    *this=UnShuffled;
}

Shuffle::ShuffleString()
{
    apstring inpString, currShuffle;
    cout << "What is the input string?" << endl;
    cin >> inpString;
    while (inpString.length()!=0)
    {
        currShuffle=inpString.substr(0,1);
        inpString=inpString.substr(1,inpString.length()-1);
        switch (*currShuffle.c_str())
        {
```

```
        case 'I':
            InShuffle();
            break;
        case 'i':
            InverseInShuffle();
            break;
        case 'O':
            OutShuffle();
            break;
        case 'o':
            InverseOutShuffle();
            break;
        }
    }
}

Shuffle::ShowCycles()
{
    for (int a=0;a<decksize;a++)
        myDeck[a].checked=false;
    int elementpos=-1;
    int i=0;
    for (int check=0;check<myDeck.length();check++)
    {
    if (!(myDeck[check].checked))
    {
        i=check;
        elementpos=-1;
        cout << "( " ;
        while (elementpos!=i)
        {
            myDeck[i].checked=true;
            for (int j=0;j<myDeck.length();j++)
            {
                if (myDeck[j].element==i)
                {
                    elementpos=j;
                    cout << myDeck[elementpos].element << " ";
```

```
                    break;
                }
            }
            i=elementpos;
            elementpos=check;
        }
        cout << ") ";
    }
    }
    cout << endl;
}

Shuffle::SetCycles()
{
    int rows,cols=0;
    for (int a=0;a<decksize;a++)
        myDeck[a].checked=false;
    int elementpos=-1;
    int i=0;
    for (int check=0;check<myDeck.length();check++)
    {
    if (!(myDeck[check].checked))
    {
        i=check;
        elementpos=-1;
        //cout << "( " ;
        while (elementpos!=i)
        {
            myDeck[i].checked=true;
            for (int j=0;j<myDeck.length();j++)
            {
                if (myDeck[j].element==i)
                {
                    elementpos=j;
                    myCycles[rows][cols] = myDeck[elementpos].element;
                    cols++;
                    break;
                }
```

```
            }
            i=elementpos;
            elementpos=check;
        }
        rows++;
        cols=0;
    }
    }
    for (int numrows=0;numrows<myCycles.numrows();numrows++)
    {
        for (int numcols=0;numcols<myCycles.numcols();numcols++)
            cout << myCycles[numrows][numcols] << ' ';
        cout << endl;
    }
}

//SHUFFLEMAIN.CPP ****** main ''client'' file

#include "shuffle.h"

int main()
{
    int size,choice;
    cout << "What size is the deck?" << endl;
    cin >> size;
    Shuffle theShuffle(size);
    while (1)
    {
        cout << "Would you like to:" << endl << endl;
        cout << "1. Change the number of shuffles/deck size" << endl;
        cout << "2. Reset the deck" << endl;
        cout << "3. In Shuffle" << endl;
        cout << "4. Out Shuffle" << endl;
        cout << "5. Inverse Out Shuffle" << endl;
        cout << "6. Inverse In Shuffle" << endl;
        cout << "7. Enter a Shuffle String" << endl;
        cout << "8. Exit" << endl;
        cin >> choice;
```

```
        switch (choice)
        {
        case 1:
            cout << "How many in the deck?" << endl;
            cin >> size;
            theShuffle.ChangeDeckSize(size);
            theShuffle.ResizeDeck(size);
            break;
        case 2:
            theShuffle.InitDeck();
            break;
        case 3:
            theShuffle.InShuffle();
            break;
        case 4:
            theShuffle.OutShuffle();
            break;
        case 5:
            theShuffle.InverseOutShuffle();
            break;
        case 6:
            theShuffle.InverseInShuffle();
            break;
        case 7:
            theShuffle.ShuffleString();
            break;
        case 8:
            return 0;
            break;
        default:
            cout << "That is not a valid choice." << endl;
            break;
        }
    }
    return 0;
}
```

Enumeration of A Shuffle Group $2 * 8 = 2^4 = 16$

Note: elements of B_n are listed as simply n. Example: $1=B_1$
A space between shuffles indicates a new conjugacy class

```
I's/O's canonical  deck
  cycle structure

IOOO    1       8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7
(0 8)(1 9)(2 10)(3 11)(4 12)(5 13)(6 14)(7 15)
OIOO    2       4 5 6 7 0 1 2 3 12 13 14 15 8 9 10 11
(0 4)(1 5)(2 6)(3 7)(8 12)(9 13)(10 14)(11 15)
OOIO    3       2 3 0 1 6 7 4 5 10 11 8 9 14 15 12 13
(0 2)(1 3)(4 6)(5 7)(8 10)(9 11)(12 14)(13 15)
OOOI    4       1 0 3 2 5 4 7 6 9 8 11 10 13 12 15 14
(0 1)(2 3)(4 5)(6 7)(8 9)(10 11)(12 13)(14 15)

IIOO    12      12 13 14 15 8 9 10 11 4 5 6 7 0 1 2 3
(0 12)(1 13)(2 14)(3 15)(4 8)(5 9)(6 10)(7 11)
IOOI    14      9 8 11 10 13 12 15 14 1 0 3 2 5 4 7 6
(0 9)(1 8)(2 11)(3 10)(4 13)(5 12)(6 15)(7 14)
OIIO    23      6 7 4 5 2 3 0 1 14 15 12 13 10 11 8 9
(0 6)(1 7)(2 4)(3 5)(8 14)(9 15)(10 12)(11 13)
OOII    34      3 2 1 0 7 6 5 4 11 10 9 8 15 14 13 12
(0 3)(1 2)(4 7)(5 6)(8 11)(9 10)(12 15)(13 14)

IOIO    13      10 11 8 9 14 15 12 13 2 3 0 1 6 7 4 5
(0 10)(1 11)(2 8)(3 9)(4 14)(5 15)(6 12)(7 13)
OIOI    24      5 4 7 6 1 0 3 2 13 12 15 14 9 8 11 10
(0 5)(1 4)(2 7)(3 6)(8 13)(9 12)(10 15)(11 14)

IIIO    123     14 15 12 13 10 11 8 9 6 7 4 5 2 3 0 1
(0 14)(1 15)(2 12)(3 13)(4 10)(5 11)(6 8)(7 9)
IIOI    124     13 12 15 14 9 8 11 10 5 4 7 6 1 0 3 2
(0 13)(1 12)(2 15)(3 14)(4 9)(5 8)(6 11)(7 10)
IOII    134     11 10 9 8 15 14 13 12 3 2 1 0 7 6 5 4
(0 11)(1 10)(2 9)(3 8)(4 15)(5 14)(6 13)(7 12)
OIII    234     7 6 5 4 3 2 1 0 15 14 13 12 11 10 9 8
(0 7)(1 6)(2 5)(3 4)(8 15)(9 14)(10 13)(11 12)
```

```
IIII    1234     15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0       0        0 8 1 9 2 10 3 11 4 12 5 13 6 14 7 15
(0)(1 2 4 8)(3 6 12 9)(5 10)(7 14 13 11)(15)
OIIOO   012      6 14 7 15 4 12 5 13 2 10 3 11 0 8 1 9
(0 12 5 6)(1 14)(2 8 13 7)(3 10 9 15)(4)(11)
OIOIO   013      5 13 4 12 7 15 6 14 1 9 0 8 3 11 2 10
(0 10 15 5)(1 8 11 13)(2 14 7 4)(3 12)(6)(9)
OIOOI   014      12 4 13 5 14 6 15 7 8 0 9 1 10 2 11 3
(0 9 10 12)(1 11 14 4)(2 13)(3 15 6 5)(7)(8)
OOIIO   023      3 11 2 10 1 9 0 8 7 15 6 14 5 13 4 12
(0 6 10 3)(1 4 14 11)(2)(5 12 15 9)(7 8)(13)
OOIOI   024      10 2 11 3 8 0 9 1 14 6 15 7 12 4 13 5
(0 5 15 10)(1 7 11 2)(3)(4 13 14 8)(6 9)(12)
OOOII   034      9 1 8 0 11 3 10 2 13 5 12 4 15 7 14 6
(0 3 5 9)(1)(2 7 13 8)(4 11)(6 15 12 10)(14)
OIIII   01234    15 7 14 6 13 5 12 4 11 3 10 2 9 1 8 0
(0 15)(1 13 4 7)(2 11 8 14)(3 9 12 6)(5)(10)

OIOOO   01       4 12 5 13 6 14 7 15 0 8 1 9 2 10 3 11
(0 8 9 11 15 7 6 4)(1 10 13 3 14 5 2 12)
OOIOO   02       2 10 3 11 0 8 1 9 6 14 7 15 4 12 5 13
(0 4 12 13 15 11 3 2)(1 6 8 5 14 9 7 10)
OOOIO   03       1 9 0 8 3 11 2 10 5 13 4 12 7 15 6 14
(0 2 6 14 15 13 9 1)(3 4 10 7 12 11 5 8)
I       04       8 0 9 1 10 2 11 3 12 4 13 5 14 6 15 7
(0 1 3 7 15 14 12 8)(2 5 11 6 13 10 4 9)
OIIIO   0123     7 15 6 14 5 13 4 12 3 11 2 10 1 9 0 8
(0 14 3 8 15 1 12 7)(2 10 11 9 13 5 4 6)
OIIOI   0124     14 6 15 7 12 4 13 5 10 2 11 3 8 0 9 1
(0 13 6 1 15 2 9 14)(3 11 10 8 12 4 5 7)
OIOII   0134     13 5 12 4 15 7 14 6 9 1 8 0 11 3 10 2
(0 11 12 2 15 4 3 13)(1 9 8 10 14 6 7 5)
OOIII   0234     11 3 10 2 9 1 8 0 15 7 14 6 13 5 12 4
(0 7 9 4 15 8 6 11)(1 5 13 12 14 10 2 3)

OO  OO     0       4 8 12 1 5 9 13 2 6 10 14 3 7 11 15
```

```
(0)(1 4)(2 8)(3 12)(5)(6 9)(7 13)(10)(11 14)(15)
OOIOIO  0013      10 14 2 6 11 15 3 7 8 12 0 4 9 13 1 5
(0 10)(1 14)(2)(3 6)(4 11)(5 15)(7)(8)(9 12)(13)
OOOIOI  0024      5 1 13 9 4 0 12 8 7 3 15 11 6 2 14 10
(0 5)(1)(2 13)(3 9)(4)(6 12)(7 8)(10 15)(11)(14)
OOIIII  001234    15 11 7 3 14 10 6 2 13 9 5 1 12 8 4 0
(0 15)(1 11)(2 7)(3)(4 14)(5 10)(6)(8 13)(9)(12)

OOIOOO  001       2 6 10 14 3 7 11 15 0 4 8 12 1 5 9 13
(0 8 10 2)(1 12 11 6)(3 4 9 14)(5 13 15 7)
OOOIOO  002       1 5 9 13 0 4 8 12 3 7 11 15 2 6 10 14
(0 4 5 1)(2 12 7 9)(3 8 6 13)(10 14 15 11)
IO      003       8 12 0 4 9 13 1 5 10 14 2 6 11 15 3 7
(0 2 10 8)(1 6 11 12)(3 14 9 4)(5 7 15 13)
OI      004       4 0 12 8 5 1 13 9 6 2 14 10 7 3 15 11
(0 1 5 4)(2 9 7 12)(3 13 6 8)(10 11 15 14)
OOIIIO  00123     11 15 3 7 10 14 2 6 9 13 1 5 8 12 0 4
(0 14 5 11)(1 10 4 15)(2 6 7 3)(8 12 13 9)
OOIIOI  00124     7 3 15 11 6 2 14 10 5 1 13 9 4 0 12 8
(0 13 10 7)(1 9 11 3)(2 5 8 15)(4 12 14 6)
OOIOII  00134     14 10 6 2 15 11 7 3 12 8 4 0 13 9 5 1
(0 11 5 14)(1 15 4 10)(2 3 7 6)(8 9 13 12)
OOOIII  00234     13 9 5 1 12 8 4 0 15 11 7 3 14 10 6 2
(0 7 10 13)(1 3 11 9)(2 15 8 5)(4 6 14 12)

OOIIOO  0012      3 7 11 15 2 6 10 14 1 5 9 13 0 4 8 12
(0 12 15 3)(1 8 14 7)(2 4 13 11)(5 9 10 6)
OOIOOI  0014      6 2 14 10 7 3 15 11 4 0 12 8 5 1 13 9
(0 9 15 6)(1 13 14 2)(3 5 12 10)(4 8 11 7)
OOOIIO  0023      9 13 1 5 8 12 0 4 11 15 3 7 10 14 2 6
(0 6 15 9)(1 2 14 13)(3 10 12 5)(4 7 11 8)
II      0034      12 8 4 0 13 9 5 1 14 10 6 2 15 11 7 3
(0 3 15 12)(1 7 14 8)(2 11 13 4)(5 6 10 9)

OOOIOOO 0001      1 3 5 7 9 11 13 15 0 2 4 6 8 10 12 14
(0 8 12 14 15 7 3 1)(2 9 4 10 13 6 11 5)
IOO     0002      8 10 12 14 0 2 4 6 9 11 13 15 1 3 5 7
(0 4 6 7 15 11 9 8)(1 12 2 5 14 3 13 10)
```

```
OIO     0003      4 6 0 2 12 14 8 10 5 7 1 3 13 15 9 11
(0 2 3 11 15 13 12 4)(1 10 7 9 14 5 8 6)
OOI     0004      2 0 6 4 10 8 14 12 3 1 7 5 11 9 15 13
(0 1 9 13 15 14 6 2)(3 8 5 11 12 7 10 4)

OOO     OOO       0 2 4 6 8 10 12 14 1 3 5 7 9 11 13 15
(0)(1 8 4 2)(3 9 12 6)(5 10)(7 11 13 14)(15)
OOOIIOO 00012     9 11 13 15 1 3 5 7 8 10 12 14 0 2 4 6
(0 12 10 9)(1 4 14 11)(2 13)(3 5 6 15)(7)(8)
OOOIOIO 00013     5 7 1 3 13 15 9 11 4 6 0 2 12 14 8 10
(0 10 15 5)(1 2 11 7)(3)(4 8 14 13)(6 9)(12)
OOOIOOI 00014     3 1 7 5 11 9 15 13 2 0 6 4 10 8 14 12
(0 9 5 3)(1)(2 8 13 7)(4 11)(6 10 12 15)(14)
IIO     00023     12 14 8 10 4 6 0 2 13 15 9 11 5 7 1 3
(0 6 5 12)(1 14)(2 7 13 8)(3 15 9 10)(4)(11)
IOI     00024     10 8 14 12 2 0 6 4 11 9 15 13 3 1 7 5
(0 5 15 10)(1 13 11 8)(2 4 7 14)(3 12)(6)(9)
OII     00034     6 4 2 0 14 12 10 8 7 5 3 1 15 13 11 9
(0 3 10 6)(1 11 14 4)(2)(5 9 15 12)(7 8)(13)
OOOIIII 0001234   15 13 11 9 7 5 3 1 14 12 10 8 6 4 2 0
(0 15)(1 7 4 13)(2 14 8 11)(3 6 12 9)(5)(10)

OOOO    OOOO      0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(0)(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)(15)

OOOIIIO 000123    13 15 9 11 5 7 1 3 12 14 8 10 4 6 0 2
(0 14 9 2 15 1 6 13)(3 7 5 4 12 8 10 11)
OOOIIOI 000124    11 9 15 13 3 1 7 5 10 8 14 12 2 0 6 4
(0 13 3 4 15 2 12 11)(1 5 7 6 14 10 8 9)
OOOIOII 000134    7 5 3 1 15 13 11 9 6 4 2 0 14 12 10 8
(0 11 6 8 15 4 9 7)(1 3 2 10 14 12 13 5)
III     000234    14 12 10 8 6 4 2 0 15 13 11 9 7 5 3 1
(0 7 12 1 15 8 3 14)(2 6 4 5 13 9 11 10)
```

Bibliography

[Ansh93] Anshel, Michael and Anshel, Iris Lee, From the Post-Markov theorem through decision problems to public-key cryptography, American Mathematical Monthly, Vol. 100, No. 9, November 1993, 835-844.

[Ansh99] Iris Anshel, Michael Anshel and Dorian Goldfeld, An algebraic method for public-key cryptography, Mathematical Research Letters 6, 1999, 287-291.

[Cox65] Coxeter, H.S.M. and Moser, W.O.J., *Generators and relations for discrete groups*, Springer-Verlag, Berlin-New York, 1965.

[Dia83] Diaconis, Graham, Kantor, The Mathematics of perfect shuffles, Advances in Applied Mathematics, 1983, 175-196.

[Eppl98] Epple, Moritz, Orbits of asteroids, a braid, and the first link invariant, Mathematical Intelligencer, Springer-Verlag, New York, 1998, 45-52.

[Gol96] Golomb, Solomon W., A Symmetry criterion for conjugacy in finite groups, Mathematics Magazine, 1996, 373-375.

[Mag66] Magnus, Karras, Solitar, *Combinatorial group theory*, John Wiley and Sons, NY-London-Sydney, 1966.

[Med87] Medvedoff, Steve and Morrison, Kent, Groups of perfect shuffles, Mathematics Magazine, 1987, 3-14.

[Morr98] Morris, S. Brent, *Magic tricks, card shuffling and dynamic computer memories*, The Mathematical Association of America, Washington D.C., 1998.

[Ram96] Ramnath, Sarnath and Scully, Daniel, Moving card i to position j with perfect shuffles, Mathematics Magazine, Vol. 69, No. 5, December 1996, 361-365.

[Rons83] Ronse, Christian, A Generalization of the perfect shuffle, Elsevier Science Publishers, 1983, 293-306.

[Rot65] Rotman, Joseph J., *The Theory of groups: an introduction*, Allyn and Bacon, Inc., Boston, Massachussetts, 1965.

[Schn96] Schneier, Bruce, *Applied cryptography*, 2nd edition, John Wiley and Sons, Inc., 1996.

[Stin95] Stinson, Douglas, *Cryptography: Theory and Practice*, CRC Press, Boca Raton, Florida, 1995.

[Wag84] Wagner, Neal R. and Magyarik, Marianne R., *Lecture Notes in Computer Science no. 196*, Springer-Verlag, 1984.

[Welsh93] Welsh, D.J.A., Complexity: knots, colourings and counting, London Mathematical Society Lecture Notes Series, No. 186, Cambridge University Press, Cambridge, Great Britain, 1993.