Cryptographic Schemes based on Rubik's Cubes

by

Tobias Nießen

BSc in Computer Science, Leibniz Universität Hannover, 2018
Master of Computer Science, University of New Brunswick, 2021

Matriculation number 3222880

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in Computer Science

At the Fakultät für Elektrotechnik und Informatik

First examiner:	Prof. Dr. rer. nat. Heribert Vollmer Institute of Theoretical Computer Science
Second examiner:	Dr. rer. nat. habil. Arne Meier Institute of Theoretical Computer Science
Advisor:	M. Sc. Sabrina Gaube Institute of Theoretical Computer Science

LEIBNIZ UNIVERSITÄT HANNOVER

September, 2021

© Tobias Nießen, 2021

Abstract

Ernő Rubik's *Magic Cube* is one of the best-selling puzzles in the world. Its structure is remarkably simple and yet, the puzzle is astonishingly difficult to solve. Aside from the challenge of physically solving the Rubik's cube, it has also attracted interest from mathematicians. Solving the puzzle means finding a sequence of permutations such that their composition moves each element of the Rubik's cube to its original position. Based on a mathematical model of Rubik's cubes, this thesis discusses difficult computational problems and how they relate to Rubik's cubes and to other computational decision and search problems. A zero-knowledge protocol that is based on the intractability assumption of the problem of solving Rubik's cubes in a fixed number of turns will be discussed. We will suggest improved parameters and a software implementation. Lastly, we develop practical attacks against a symmetric encryption scheme that is based on Rubik's cubes, which effectively break the scheme's security almost entirely.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Der Mensch spielt nur, wo er in voller Bedeutung des Wortes Mensch ist, und er ist nur da ganz Mensch, wo er spielt.
— Schiller, *Über die ästhetische Erziehung des Menschen*

In 1974, Ernő Rubik, an Hungarian professor of architecture, invented a puzzle, which he called his *Magic Cube*. To him, it was an architectural challenge to design the cube in such a way that its elements could move around all three axes, and the difficulty of physically constructing such a cube was intriguing. Once he began marking the faces of the cube with different colors, he himself was surprised about the difficulty of restoring the cube to its original color pattern. Despite this realization, Rubik remained more interested in designing and constructing the puzzle than in solving it [56].
Since its invention, the *Rubik's cube* has been one of the best-selling toys in the world. Its structure is remarkably simple and yet, the puzzle is astonishingly difficult to solve. The fact that such a simple toy could allow a total of $43,252,003,274,489,856,000$ possible positions may even seem strange. Of course, Rubik and others were quick to design larger cubes consisting of more movable elements, such as the 4x4x4 Rubik's cube ("Rubik's Revenge" or "Master Cube") and the 5x5x5 Rubik's cube ("Professor's Cube"), and the number of possible positions grows exponentially with the size of the cube.

Over the last decades, the puzzle became an object of interest in different areas. Most importantly, of course, the puzzle remains a challenge for humans. While various methods are known for solving the cube, it is inherent to the human nature to strive to become better. There are competitions, rankings, and national and international records for solving a Rubik's cube as quickly as possible, which is referred to as *speedcubing*. In most competitions, contestants perform dozens of turns within mere seconds to solve the cube in as little time as they can. Nevertheless, there are also competitions that challenge contestants to solve the cube in as few turns as possible, or to solve it as quickly as possible with one hand only, or even to solve the cube while blindfolded [72].

Aside from the challenges of physically solving the Rubik's cube, it has also attracted interest from mathematicians. While, from an architectural perspective, the geometry of the three-dimensional cube and the movement of its elements might have been inspiring, the cube turned out to have remarkable algebraic properties as well. Within the puzzle, each move is a permutation of the positions of the colored facelets. Solving the puzzle means finding a sequence of such permutations such that their composition moves each facelet to its original position. The permutations that represent turns of the faces of the Rubik's cube generate a permutation group, and this group is a an exact model of all possible positions and all possible moves within the puzzle [56, pp. 155-167].

Based on this understanding of the Rubik's cube, many questions about the puzzle could now be answered algebraically without the physical embodiment of the cube. However, just like the physical Rubik's cube, this group surprised with its enormous complexity. One of the most interesting problems related to Rubik's cubes was the search for *God's number*, that is, the maximum number of moves that an optimal algorithm would require to solve an arbitrary position of the Rubik's cube. Because it was believed that no human would be able to find an optimal solution to arbitrary positions, such an algorithm is commonly referred to as *God's algorithm*. Mathematically, God's number is the *diameter* of the Rubik's cube permutation group. Despite significant efforts, God's number remained unknown for decades, until, in 2010, vast computational resources were used to determine the permutation group's diameter [54, 55]. Still, this computer-assisted proof did not yield optimal solutions for all possible positions, but only proved a tight upper bound. Finding an optimal solution efficiently remains an unsolved problem.

Of course, seemingly difficult problems attract interest from cryptographers, and the Rubik's cube is no exception. Based on the mathematical model of Rubik's cubes that is defined in Chapter 2, this thesis discusses such problems and how they relate to Rubik's cubes and to other computational decision and search problems in Chapter 3. In Chapter 4, we will discuss a zero-knowledge protocol that is based on the intractability assumption of the problem of solving Rubik's cubes in a fixed number of turns, and, in Chapter 5, we will discuss and then break a recently proposed symmetric encryption scheme whose security is supposedly based on the intractability assumption of the conjugacy problem for the Rubik's cube permutation group. Finally, in Chapter 6, we will briefly list other proposals for cryptographic methods that are related to Rubik's cubes.

In particular, aside from surveying existing work, this thesis contributes

- new and presumably more efficient parameter sets for the zero-knowledge protocol in Chapter 4 (or, alternatively, parameter sets offering a higher degree of security at the same level of efficiency),
- a reference implementation of the zero-knowledge protocol in Python,
- an optimized implementation of the zero-knowledge protocol in C that focuses on the prevention of side-channel attacks, and
- the first cryptanalysis of a recently proposed symmetric encryption scheme in Chapter 5, including practical attacks, ranging from attacks that disprove IND-CPA security to partial plaintext recovery, ciphertext forgery, and potentially even key recovery attacks, as well as an extension of the discovered weaknesses to recently proposed commitment schemes.

Chapter 2

Mathematical representation

I respect the cube. I cannot fathom it. Whichever way I turn, disorder gives way to more disorder.

— György Marx, *Rubik's Cubic Compendium*

Rubik's cubes are a shining example of group theory and permutation groups. This chapter defines the mathematical model that accurately represents the properties of Rubik's cubes of almost arbitrary sizes.

2.1 Singmaster notation

Any cube has six faces, twelve edges, and eight corners. A Rubik's cube, however, divides each face and each edge into smaller parts. The smaller cubes that it appears to consist of are called *cubies*, and the visible faces of these smaller cubes are called *facelets*.

Cubies	$6 \cdot (n-1)^2 + 2$
Corner cubies	8
Edge cubies	$12 \cdot (n-2)$
Center cubies	$6 \cdot (n-2)^2$
Facelets per face	n^2
Facelets	$6n^2$

Table 2.1: Basic structural properties of Rubik's cubes of size n

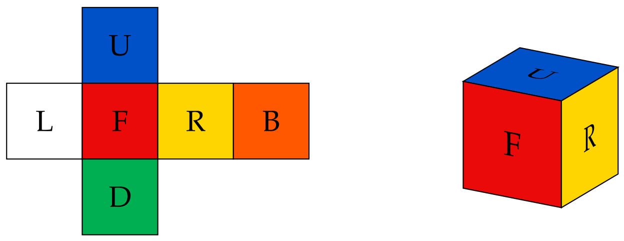


Figure 2.1: Labels assigned to the faces of a Rubik's cube

Each face of a Rubik's cube has its own color. The positions of the colored faces on a solved Rubik's cube, relative to each other, cannot be changed. However, the positions of the colored faces, relative to the observer, can change, and, in the more general case of Rubik's cubes whose faces do not have center facelets (e.g., the 4x4x4 cube), it can even be impossible to determine the "correct" color of a face unambiguously.

Therefore, it is convenient to identify the faces of the Rubik's cube based on their position relative to the observer: front (F), back (B), left (L), right (R), up (U), and down (D). Figure 2.1 shows the initial state of a Rubik's cube of unspecified size. The assignment of colors to the faces is arbitrary, but commonly used. [56, p. 28].

Figure 2.1 also serves to demonstrate the projection of the faces of the three-dimensional cube to a two-dimensional plane that will be used throughout this thesis due to the constraints of two-dimensional media.

Remark 2.1. Some mathematicians prefer *top* (T) over *up* (U) [56, p. 26].

The color of each facelet is determined by its initial position on the cube.

The standard approach to describing a sequence of moves is the Singmaster notation [66]. Since a move is always the rotation of one side of the cube, it is sufficient to note which face is being rotated, and by what angle, where the angle is a multiple of a clockwise quarter turn. We use the same letters that we assigned to the faces of the cube to denote the turns. For example, the sequence FFR means that F is rotated by a clockwise quarter turn twice, followed by R being rotated by a clockwise turn once. Repeating the same move multiple times can be abbreviated using exponents: F^2R. Similarly, negative exponents mean counterclockwise turns, and since one counterclockwise turn is equivalent to three clockwise turns, the sequences F^{-1} and F^3 are interchangeable in terms of the outcome. Similarly, sequences of moves can be repeated as well, e.g., $(FR)^2 = FRFR$ [56, p. 27].

2.2 Permutation-based representation

In this section, we will develop the representation of Rubik's cubes as permutation groups. Recall that permutations over a domain D are bijective functions $\sigma\colon D \to D$. For simplicity, we will only use domains that consist of the first n natural numbers for some n. Consequently, any permutation of n elements is a bijective function $\sigma\colon \{1, \dots, n\} \to \{1, \dots, n\}$. S_n denotes the *symmetric group*, which is the group whose underlying set is the set of all permutations of n elements and whose group operation is function composition. Subgroups of symmetric groups are called *permutation groups*. In particular, if $\mathcal{S} \subseteq S_n$ is a set of permutations from some symmetric group S_n, then $\langle \mathcal{S} \rangle$ is the subgroup of S_n whose generators are the elements of $\mathcal{S}$, i.e., $\langle \mathcal{S} \rangle \leq S_n$. The order $|G|$ of a group G is the cardinality of the underlying set, for example, $|S_n| = n!$ for $n \in \mathbb{N}$. The order of an element σ of the group is the order of its cyclic subgroup $\langle \sigma \rangle$.

Remark 2.2. We use the conventional notation $f \circ g$ for function composition from right to left, that is, $(f \circ g)(x) = f(g(x))$. Because permutations are functions, the composition of permutations is *usually* applied from right to left. However, in literature related to permutation groups, the group operation is commonly applied from left to right (refer to modern standard works on permutation groups, such as Dixon-Mortimer [17, p. 3] or Cameron [7, p. 2]), and this rule is also widely supported by mathematicians who have had a major impact on our understanding of the Rubik's cube, such as David Singmaster [66, p. 5]. Following that convention, we use the notation $\sigma_1\sigma_2$ to denote the group operation $\sigma_1\sigma_2 := \sigma_2 \circ \sigma_1$. We generally avoid the notation $\sigma_1 \cdot \sigma_2$ (or $\sigma * \sigma_2$) in an attempt to not further add to the confusion, albeit we use $*$ for the composition of permutations in SageMath [68] code, where it also composes from left to right. Be aware that some related work uses the same notation $\sigma_1\sigma_2$ to denote $\sigma_2 \circ \sigma_1$, but additionally defines $\sigma_1 * \sigma_2$ as a

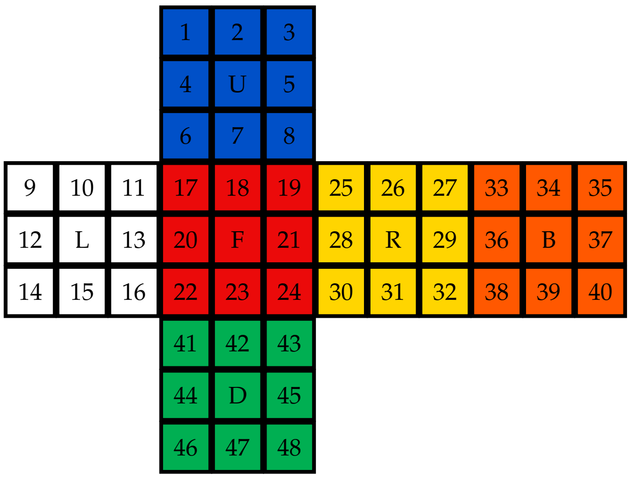


Figure 2.2: Numbers assigned to the facelets of a 3x3x3 Rubik's cube

right group action that is equivalent to $\sigma_1 \circ \sigma_2$ and thus means the opposite of the notation $\sigma_1 * \sigma_2$ in SageMath [68].

The state of a standard[1] Rubik's cube of arbitrary size is defined in its entirety by the position of each facelet on the surface of the cube. We assign a number to each facelet. The standard numbering of the facelets of a 3x3x3 Rubik's cube is shown in Figure 2.2. Each possible move is a rotation of a face of the cube, which we represent as permutations of the facelets (or, more accurately, as permutations of the positions of the facelets). The standard generators for the 3x3x3 Rubik's cube are the following permutations.

$$
\begin{aligned}
U &:= (1\ 3\ 8\ 6)\,(2\ 5\ 7\ 4)\,(9\ 33\ 25\ 17)\,(10\ 34\ 26\ 18)\,(11\ 35\ 27\ 19) \\
L &:= (9\ 11\ 16\ 14)\,(10\ 13\ 15\ 12)\,(1\ 17\ 41\ 40)\,(4\ 20\ 44\ 37)\,(6\ 22\ 46\ 35) \\
F &:= (17\ 19\ 24\ 22)\,(18\ 21\ 23\ 20)\,(6\ 25\ 43\ 16)\,(7\ 28\ 42\ 13)\,(8\ 30\ 41\ 11) \\
R &:= (25\ 27\ 32\ 30)\,(26\ 29\ 31\ 28)\,(8\ 33\ 48\ 24)\,(5\ 36\ 45\ 21)\,(3\ 38\ 43\ 19) \\
B &:= (33\ 35\ 40\ 38)\,(34\ 37\ 39\ 36)\,(3\ 9\ 46\ 32)\,(2\ 12\ 47\ 29)\,(1\ 14\ 48\ 27) \\
D &:= (41\ 43\ 48\ 46)\,(42\ 45\ 47\ 44)\,(14\ 22\ 30\ 38)\,(15\ 23\ 31\ 39)\,(16\ 24\ 32\ 40)
\end{aligned}
$$

[1]There are variants of the Rubik's cube that mark some or all facelets with symbols. The orientation of those symbols is not reflected by the permutation-based model, therefore, we restrict ourselves to *standard* Rubik's cubes.

We do not assign numbers to the center facelets. Because their position is not affected by rotating any of the faces, even if we assigned numbers to these facelets, all moves would map these numbers to themselves, and the order of the permutation group would remain the same. Conversely, we could allow rotating the center slices around each axis, which would change the positions of the center facelets, however, this would unnecessarily complicate the representation and not create any interesting properties. Instead, we assume that the center facelets are fixed in space.

Definition 2.3. The *Rubik's cube group* (*for the 3x3x3 Rubik's cube*) is the permutation group

$$\mathfrak{R}_3 := \langle \{ U, L, F, R, B, D \} \rangle < S_{48}.$$

With this definition, the Singmaster notation (see Section 2.1), which we introduced as a mere way of writing sequences of turns, is equivalent to the mathematical notation. The sequence F^2R means that the front face is rotated by $180\,\mathrm{deg}$ before turning the right face by $90\,\mathrm{deg}$. Since the group operation is function composition, and because F and R are permutations and thus functions, $\sigma = F^2R$ means that $\sigma(i) = R(F(F(i)))$, which is the desired effect. This is a convenient consequence of the left-to-right notation that we use for the group operation (see Remark 2.2).

The next chapters will also use larger Rubik's cubes, especially the 5x5x5 Rubik's cube, which is often referred to as the Professor's cube. Ernő Rubik himself referred to it as "the giant cube" [56, p. 13]. Again, we assign numbers to all facelets except the centers of the six faces (see Figure 2.3) and define moves via permutations. However, in addition to allowing turning the faces of the cube, a 5x5x5 cube also allows turning the layer behind each face, which leads to a total of twelve possible moves at each point. Mathematically, the permutation group representing the 5x5x5 Rubik's cube is a subset of S_{144} and has twelve generating permutations. Due to the complexity of these permutations, they are listed separately in Appendix A.

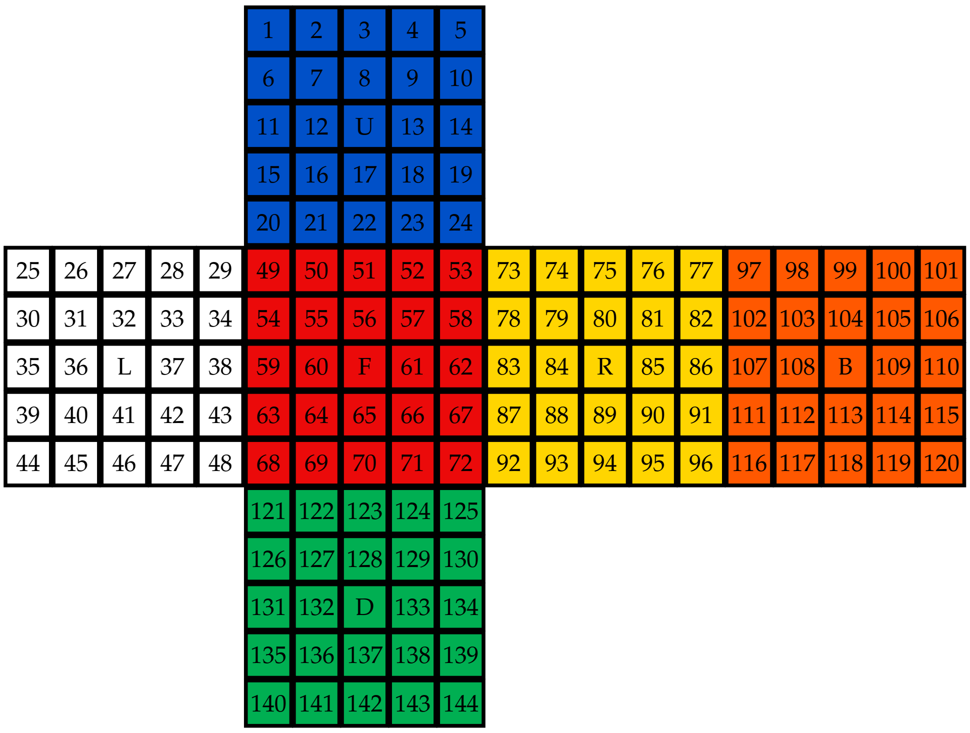


Figure 2.3: Numbers assigned to the facelets of a 5x5x5 Rubik's cube

Definition 2.4. The *Rubik's cube group for the 5x5x5 Rubik's cube* is the permutation group

$$\mathfrak{R}_5 := \langle \{ U_0, U_1, L_0, L_1, F_0, F_1, R_0, R_1, B_0, B_1, D_0, D_1 \} \rangle < S_{144}.$$

In general, we will denote with an index the distance of the layer from the respective face, i.e., Q_d is a clockwise quarter turn of the layer at depth $d \geq 0$ from the face $Q \in \{ U, L, F, R, B, D \}$, where $d = 0$ is used for the face itself, which is the outermost layer, and $d > 0$ for the slices between the face and the center of the cube. The maximum depth for a cube with edge length n is $\lfloor n/2 \rfloor - 1$ and since there are six faces, the *standard set of generators* for a Rubik's cube of size n consists of $6 \cdot \lfloor n/2 \rfloor$ permutations, which represent quarter turns of single layers. For the 2x2x2 and 3x3x3 Rubik's cubes, we omit the index since only one generator exists per face. The 4x4x4 and 5x5x5 Rubik's cubes have two movable layers for each face, which results in a total of twelve generators. The 6x6x6 and 7x7x7 Rubik's cubes have three movable layers per face and thus a total of 18 generators.

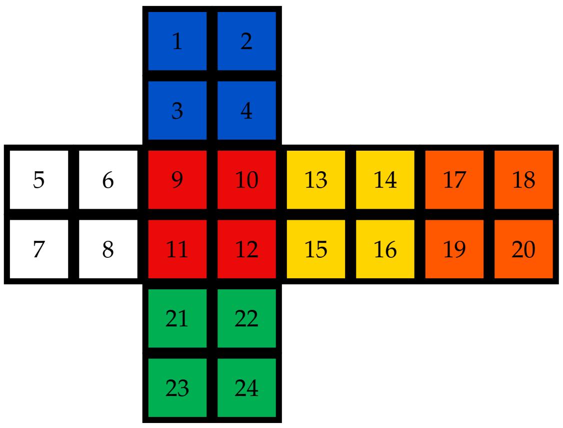


Figure 2.4: Numbers assigned to the facelets of a 2x2x2 Rubik's cube

An example of a Rubik's cube whose edges have even length is the 2x2x2 Rubik's cube, which is the smallest meaningful (cubic) construction (see Figure 2.4). The 3x3x3 and the 2x2x2 Rubik's cubes are the only Rubik's cubes that can be represented using only six generators each. Observe that the 2x2x2 Rubik's cube has no center facelets, which means that no facelets are fixed in space. Consequently, when solved, the cube might appear to have been rotated in space. For example, after a single F turn, the 2x2x2 Rubik's cube can be solved either by undoing the first turn, i.e., through F^{-1}, or by turning the back face to match the rotation of the front face, i.e., through B^{-1}. The first option restores the initial state, the second appears to have rotated the cube in space by a clockwise quarter turn around the axis going through the front and back face, which means that the upper face is now white, not blue.

2.3 Turn metrics

All moves can be constructed from the generators of the permutation group. For example, a half turn $(180°)$ of the front face can be expressed as F^2. However, there is no universal agreement whether this sequence should be considered to be a single move or not. When holding a Rubik's cube, it certainly seems like a single move, on the other hand, it can clearly be constructed from two smaller moves.

Similarly, a counterclockwise turn of any face requires three moves in the previously defined mathematical model since F^{-1} is not a generator of the permutation group, and the shortest factorization of F^{-1} using generating permutations is F^3. However, intuitively, a counterclockwise turn surely is a single move.

More ambiguities arise for larger Rubik's cubes, which have more than one rotatable layer per face. For example, turning both F layers of a 5x5x5 Rubik's cube can be written as F_0F_1 (or F_1F_0), however, the physical constructions of such cubes generally allow turning both layers in a single move. In fact, rotating only the inner layer is often not considered a single move in reality since it is simpler to first rotate both the outer and inner layer (F_0F_1) before undoing the turn of the outer layer (F_0^{-1}). In the previously defined standard set of generators, this simple sequence consists of five clockwise quarter turns while being equivalent to the single clockwise quarter turn sequence F_1.

To solve these ambiguities, turn metrics can be defined.

Definition 2.5. A *turn metric* for a Rubik's cube of size n is a set $T \subset \mathfrak{R}_n$ with $\mathfrak{R}_n = \langle T \rangle$ and such that each element of T is the composition of commuting generators from the standard set of generators for $\mathfrak{R}_n$.

Example. The set $\{ UR, LR, FR, R, BR, DR \}$ generates $\mathfrak{R}_3$ but not all of its elements are compositions of commuting permutations from the standard set of generators for $\mathfrak{R}_3$. Intuitively, the moves UR, FR, BR, and DR cannot be executed as a single turn since the respective layers cannot be turned simultaneously.

For the standard 3x3x3 Rubik's cube, the most common metrics are the half turn metric and the quarter turn metric.

Definition 2.6 (HTM)**.** The *half turn metric* is the turn metric

$$\{ f^x \mid f \in \{ U, L, F, R, B, D \}, x \in \{ -1, 1, 2 \} \}.$$

Definition 2.7 (QTM)**.** The *quarter turn metric* is the turn metric

$$\{f^x \mid f \in \{U, L, F, R, B, D\}, x \in \{-1, 1\}\}.$$

Just like the permutation-based model that was defined in Section 2.2, neither the HTM nor the QTM allows turning the center slices of the cube. However, this restriction is lifted by the slice turn metric and quarter slice turn metric. In the permutation-based model, the center slices are assumed to be fixed, therefore, turning them is equivalent to turning the two adjoining layers in opposite directions.

Definition 2.8 (STM)**.** The *slice turn metric* is the turn metric

$$\{f^x \mid f \in \{U, L, F, R, B, D, D^{-1}U, L^{-1}R, F^{-1}B\}, x \in \{-1, 1, 2\}\}.$$

The half turn metric (HTM), the quarter turn metric (QTM), and the slice turn metric (STM) were described, for example, by Gerzson Kéri in 1987 [56, p. 97].

Definition 2.9 (QSTM)**.** The *quarter slice turn metric* is the turn metric

$$\{f^x \mid f \in \{U, L, F, R, B, D, D^{-1}U, L^{-1}R, F^{-1}B\}, x \in \{-1, 1\}\}.$$

For Rubik's cubes of any size, the outer block turn metric (OBTM) is commonly used. This metric is used, for example, by the World Cube Association at competitions [73].

Definition 2.10 (OBTM)**.** Let n be the size of the Rubik's cube. The outer block $\mathcal{B}_{Q,d}$ for a face $Q \in \{U, L, F, R, B, D\}$ and a depth $0 \leq d < \lfloor \frac{n}{2} \rfloor$ is the rotation $Q_0Q_1Q_2 \ldots Q_d$, that is, the simultaneous clockwise quarter turn of the $d + 1$ outermost layers, beginning with the face Q. The *outer block turn metric* is the turn metric

$$\left\{\mathcal{B}^x_{Q,d} \;\middle|\; Q \in \{U, L, F, R, B, D\}, d \in \left\{0, 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor - 1\right\}, x \in \{-1, 1, 2\}\right\}.$$

For $n = 3$, OBTM is equivalent to the half turn metric (see Definition 2.6).

To simplify operations within the mathematical model, we use the following turn metric that is a natural consequence of the definition of the Rubik's cube group for cubes of any size. It requires $6 \cdot \lfloor n/2 \rfloor$ generators only, compared to $18 \cdot \lfloor n/2 \rfloor$ generators when using OBTM. Additionally, for $n > 3$, at least half of the OBTM generators permute multiple layers simultaneously, which means that those generators are more complex than single-layer permutations.

Definition 2.11 (CQLTM)**.** The *clockwise quarter layer turn metric* is the standard set of generators

$$\begin{gathered}\big\{ U_0, \ldots, U_{\lfloor \frac{n}{2} \rfloor - 1}, L_0, \ldots, L_{\lfloor \frac{n}{2} \rfloor - 1}, F_0, \ldots, F_{\lfloor \frac{n}{2} \rfloor - 1}, \\ R_0, \ldots, R_{\lfloor \frac{n}{2} \rfloor - 1}, B_0, \ldots, B_{\lfloor \frac{n}{2} \rfloor - 1}, D_0, \ldots, D_{\lfloor \frac{n}{2} \rfloor - 1} \big\}.\end{gathered}$$

Remark 2.12. We use the term *layer* instead of *slice* in Definition 2.11 to avoid confusion with the slice turn metric (Definition 2.8) and quarter slice turn metric (Definition 2.9). While the standard set of generators allows turning faces and slices, it does not allow turning the center slices (which only exist if n is odd).

2.4 Complexity of the Rubik's cube

The model that was developed in Section 2.2 accurately represents Rubik's cubes. Therefore, it can be used to determine certain complexity measures.

Since the order of a permutation group is the number of permutations within the group and because the state of a Rubik's cube is determined by the permutation of its facelets, the order of the Rubik's cube group is the number of possible states (or positions).

The order[2] of the 3x3x3 Rubik's cube group, $|\mathfrak{R}_3|$, is 4.3×10^{19} (43 quintillion), or 1.2×2^{65}. Each of these possibilities is unique and visually distinguishable from every other possible state. In other words, the number of possible positions is large enough to give a unique instance of the puzzle to each living person on more than five billion Earths, and none of them would have the same solution.

[2]To determine the orders of large permutation groups, the mathematical software system SageMath [68] can be used.

Before considering the orders of groups representing Rubik's cubes of other sizes, we will briefly discuss the perhaps most significant question surrounding the Rubik's cube.

Definition 2.13 (God's number)**.** The smallest number of turns that is sufficient to solve any of the possible states of a Rubik's cube is called *God's number*.

Of course, God's number depends on the turn metric that is used. We can develop a simple lower bound as follows.

Theorem 2.14. *Given a turn metric S and a state σ, there are states that cannot be reached from σ in less than $\left\lceil \log_{|S|}(|\langle S\rangle|) - 1 \right\rceil$ moves.*

Proof. Let $m \geq 0$. The number of different sequences of m moves from S is $|S^m| = |S|^m$, therefore, such sequences cannot reach more than $|S|^m$ different states.[3] Sequences of length up to m can, therefore, only reach a maximum of $s_m = \sum_{i=0}^{m} |S|^i$ different states. We have

$$s_m = \sum_{i=0}^{m} |S|^i = \frac{|S|^{m+1} - 1}{|S| - 1} \leq |S|^{m+1}.$$

The total number of states that are reachable from σ is $|\langle S\rangle|$. If $s_m < |\langle S\rangle|$, then there must exist at least one state in $\langle S\rangle$ that cannot be reached in m moves. It follows that

$$\begin{aligned} & m < \log_{|S|}(|\langle S\rangle|) - 1 \\ \implies \quad & m + 1 < \log_{|S|}(|\langle S\rangle|) \\ \implies \quad & |S|^{m+1} < |\langle S\rangle| \\ \implies \quad & s_m < |\langle S\rangle|. \end{aligned}$$

Therefore, if $m < \left\lceil \log_{|S|}(|\langle S\rangle|) - 1 \right\rceil$, the number of states that can be reached in m turns is smaller than the number of all states. □

Upon closer inspection, God's number for the 3x3x3 Rubik's cube is the *diameter* of the group $\mathfrak{R}_3$. This leads to the following conclusion.

[3] In fact, for $m \geq 2$, the number of different sequences of moves is much larger than the number of different resulting states (due to, e.g., $F^4 = R^4$). Tighter lower bounds can be achieved here and in Corollary 2.15 by counting states instead of sequences of moves.

Turn metric	HTM	QTM	STM	QSTM	CQLTM
Number of generators	18	12	27	18	6
Lower bound for God's number	15	18	13	15	25

Table 2.2: Trivial lower bounds for God's number for the 3x3x3 Rubik's cube

Corollary 2.15. *Given a turn metric S for the 3x3x3 Rubik's cube, a lower bound for God's number is* $\left\lceil \log_{|S|}(|\mathfrak{R}_3|) - 1 \right\rceil$.

Proof. The identity permutation is the only permutation in $\langle S \rangle = \mathfrak{R}_3$ that represents the 3x3x3 Rubik's cube in a solved state. Let $\sigma = \mathrm{id}_{\mathfrak{R}_3}$ be the identity permutation and let $m < \left\lceil \log_{|S|}(|\langle S \rangle|) - 1 \right\rceil$. Then, according to Theorem 2.14, there exists at least one state in $\langle S \rangle$ that cannot be reached from the solved state σ in up to m moves. Therefore, God's number must be greater than m. □

Table 2.2 shows lower bounds for God's number for the previously defined turn metrics based on the order of the group and Corollary 2.15. It does not list OBTM because OBTM is the same as HTM for 3x3x3 Rubik's cubes.

By 1980, Singmaster had proved a lower bound of 18 for God's number for the 3x3x3 Rubik's cube in the half turn metric. He added that "one is tempted to conjecture that every position can be achieved in at most 20 moves" [66, p. 34], even though the best known upper bound was 52 [66, p. 39]. Despite significant efforts, it took another 30 years for the conjecture to be proven.

In fact, for the 3x3x3 Rubik's cube, God's number in the half turn metric and in the quarter turn metric is known. In the half turn metric, God's number is 20, meaning that every position of the cube can indeed be solved in no more than 20 moves [54, 55]. In the quarter turn metric, God's number is 26 [53]. These values were obtained through computer-assisted proofs using many years of CPU time. For other turn metrics of the standard 3x3x3 Rubik's cube, and for larger cubes, God's number is still unknown.

Of course, the group order increases exponentially with the size of the Rubik's cube, and, unsurprisingly, the order of the 5x5x5 Rubik's cube group, $|\mathfrak{R}_5|$, is much larger than that of the smaller cube, and is 2.6×10^{90} (2.6 novemvigintillion), or 1.3×2^{300}. It is difficult to put this number into perspective since it is much larger than the estimated number of atoms in the observable universe.

Estimating God's number for the 5x5x5 cube is more difficult than for the 3x3x3 cube, and not only due to its greater complexity. Similar to the smaller cube, after m clockwise quarter turns of any of the twelve layers, the cube is in one of up to 12^m possible states. The order of the group is 6.9×12^{83}, therefore, there must exist states of the cube that are reachable, but in no less than 83 clockwise quarter turns. Again, there must exist states for which the same minimum number of clockwise quarter turns is required to return to the initial state. In other words, 83 is a lower bound for the diameter of $\mathfrak{R}_5$ when using the standard set of generators.

However, unlike the smaller 3x3x3 Rubik's cube, the 5x5x5 Rubik's cube does not have a unique solved state. While the smaller Rubik's cube is only solved when all facelets are in their original position, meaning that the permutation representing the state of the cube is the identity permutation, there are many states of the 5x5x5 Rubik's cube that represent the cube in a solved state.[4] In other words, some states are visually indistinguishable because the respective permutations only differ in the positions of facelets that have the same color. This is not an inaccuracy in the model since the facelet positions are truly different in reality, but the difference is usually not observed or considered. Thus, God's number for cubes other than the 3x3x3 Rubik's cube is bounded from above by the diameter of the group, but not necessarily the same number.

[4]Conversely, not every permutation from S_{144} that appears to represent a solved state is reachable from the initial state of the Rubik's cube, i.e., is not in $\mathfrak{R}_5$.

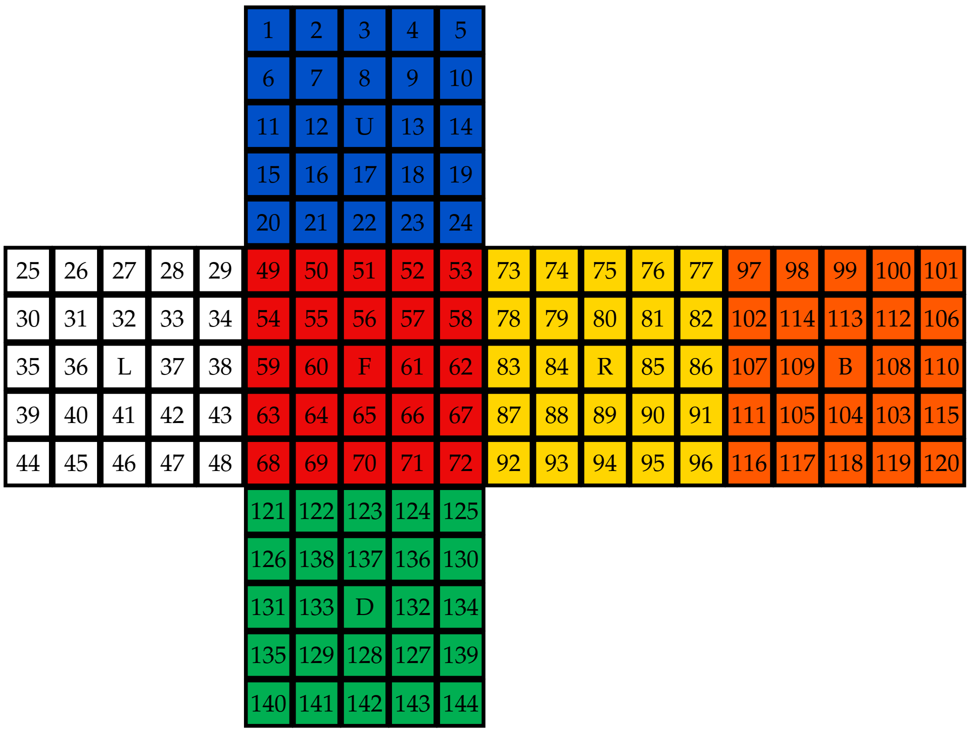


Figure 2.5: Rubik's cube in a valid, solved state that is different from the initial state

Example. Consider this sequence:

$$\begin{aligned} & L_0 R_0 U_0 U_0 R_0 L_0^{-1} B_0^2 U_0^2 R_0^2 F_0^2 L_0^2 D_0^2 L_0^2 F_0^2 \\ = & \,(103\ 114)\,(104\ 113)\,(105\ 112)\,(108\ 109) \\ & (127\ 138)\,(128\ 137)\,(129\ 136)\,(132\ 133) \end{aligned}$$

Since each cycle only contains indices of facelets of the same color, the result represents a solved state (see Figure 2.5) despite being different from the initial state.

Calculation shows that each visually distinguishable state of the 5x5x5 Rubik's cube can be represented by $(4!)^{12}/4 = 9,130,086,859,014,144$ different permutations from $\mathfrak{R}_5$. Therefore, there are only $\frac{4 \cdot |\mathfrak{R}_3|}{(4!)^{12}} \approx 2.83 \times 10^{74}$ (283 tresvigintillion) visually distinguishable states. Consequently, there are 9.13×10^{15} (9.13 quadrillion) solved states, i.e., states that are visually indistinguishable from the initial state.

Lastly, $|\mathfrak{R}_2|$, the order of the 2x2x2 Rubik's cube, is only $88,179,840$. As mentioned earlier, none of its facelets are fixed in space since the length of its edges is even, which allows rotating the entire cube. Therefore, there are $6 \cdot 4 = 24$ different orientations of the entire cube in space, and thus only $88,179,840/24 = 3,674,160$ truly different color patterns. On the other hand, there are 24 solved states. Because $\mathfrak{R}_2$ is tiny compared to the groups representing the 3x3x3 and the 5x5x5 Rubik's cubes, God's number is relatively easy to determine for the 2x2x2 Rubik's cube, and is 11 in the half turn metric and 14 in the quarter turn metric. While even this puzzle can be challenging to solve for humans, it is of little interest to us due to its low complexity. It is included here only as an example of a cube whose edges have even length.

Larger cubes of even size (e.g., 6x6x6) combine the lack of fixed-in-space facelets (as discussed for the 2x2x2 Rubik's cube) with the non-injective mapping of color patterns to permutations (as discussed for the 5x5x5 Rubik's cube).

Despite such symmetries and visually indistinguishable permutations, the number of visibly different positions still grows exponentially. In particular, Demaine et al. proved that God's number grows as $\Theta\left(n^2/\log n\right)$, where n is the size of the Rubik's cube [14].

2.5 Variegation (visual disorder)

György Marx defines and uses variegation as a metric for the degree of disorder of states of the 3x3x3 Rubik's cube [56, p. 179].

Definition 2.16 (Variegation)**.** Let $\mathcal{F} = \{ U, L, F, R, B, D \}$ be the set of faces of the Rubik's cube and let $f\colon \{ 1, \ldots, 48 \} \to \mathcal{F}$ be the surjective function that maps each facelet to the face that it was initially on. Let $\sigma \in \mathfrak{R}_3$ be a state of the Rubik's cube. The *variegation* of a face $Q \in \mathcal{F}$ in the state σ is

$$\operatorname{varg}_Q(\sigma) := 9^2 - \sum_{P \in \mathcal{F}} \left(\operatorname{orig}_{Q,P}(\sigma)\right)^2,$$

where $\operatorname{orig}_{Q,P}\colon \mathfrak{R}_3 \to \{ 1, \ldots, 9 \}$ is the number of facelets on the face Q that were origi-

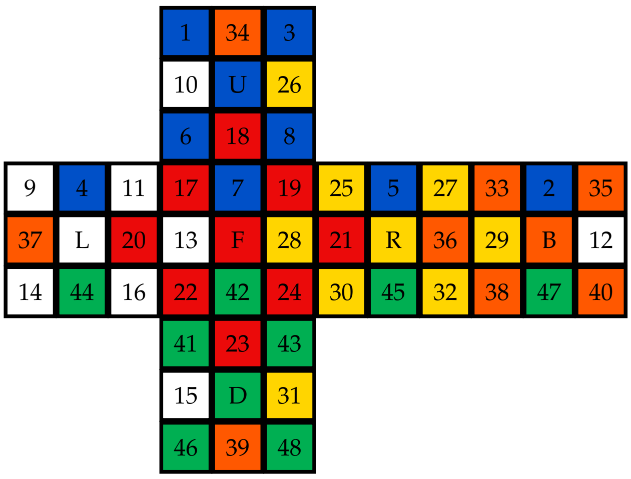


Figure 2.6: The Superflip position: all cubies are in the right position, but all edges are flipped

nally on face P (including the center of the face if $P = Q$), i.e.,

$$\mathrm{orig}_{Q,P}(\sigma) := |\{i \in \{1, \ldots, 48\} \mid f(i) = P, f(\sigma(i)) = Q\}| + \begin{cases} 1 & \text{if P = Q,} \\ 0 & \text{otherwise.} \end{cases}$$

Definition 2.17 (Average variegation)**.** Let $\mathcal{F}$ and σ be defined as in Definition 2.16. The *average variegation* of σ is

$$\mathrm{varg}(\sigma) := \frac{1}{|\mathcal{F}|} \cdot \sum_{Q \in \mathcal{F}} \mathrm{varg}_Q(\sigma).$$

Intuitively, a smaller average variegation corresponds to less disorder in the cube's state. When the cube is solved, the average variegation is zero. The variegation also allows some insight into why Rubik's cubes are difficult to solve. So far, we have seen that there is an enormous number of different states, and that some states require 20 moves to solve in the half turn metric, but, based on these properties alone, the correct choice of each move could still be obvious, in which case it would not be difficult to solve the cube after all. One particularly difficult instance of the standard Rubik's cube is the so called Superflip, which is shown in Figure 2.6. It has been proven that it is impossible to solve this state in less than 20 moves in the half turn metric, therefore, no position exists that requires more

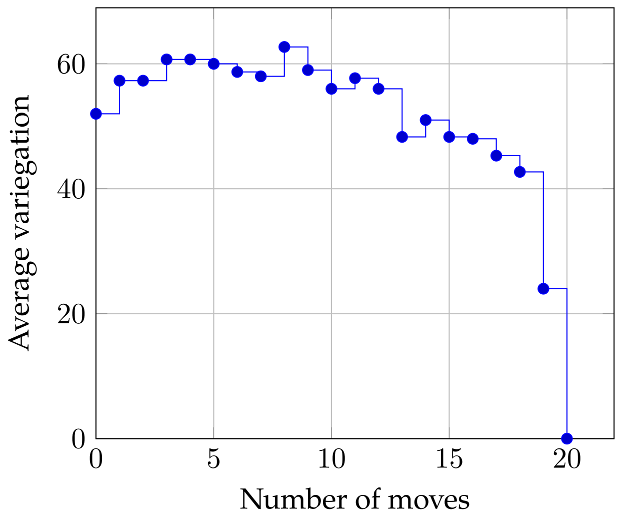


Figure 2.7: Average variegation during an optimal HTM solution of the Superflip

moves [55]. In the quarter turn metric, however, this state can be solved in 24 quarter turns, which is less than God's number for the quarter turn metric, and, therefore, more difficult states exist in the quarter turn metric [53].
An optimal solution for the Superflip in the half turn metric [28] is

$$FBU^2RF^2R^2B^2U^{-1}DFU^2R^{-1}L^{-1}UB^2DR^2UB^2U.$$

Figure 2.7 shows how the average variegation changes when applying the solution moves, one by one. The average variegation of the Superflip position itself is only 52.0. However, the first moves of the optimal solution increase the disorder further. Only after 13 moves is the average variegation lower than it initially was. If one attempted to reduce the degree of disorder from the beginning, it would, with high probability, lead to a less than optimal solution. For example, it is rarely helpful to solve single faces of the cube even though this often reduces the degree of disorder. As Figure 2.7 shows, the visual disorder does not necessarily correlate with the number of required moves to solve the cube.

2.6 Suggested literature

David Singmaster's *Notes on Rubik's Magic Cube* [66] are an excellent introduction to the mathematical model of the Rubik's cube and go far beyond what has been presented here. Of particular interest might be the subgroups of the cube group that Singmaster discusses at length, as well as early results concerning upper and lower bounds for God's number.
A more recent introductory book on the group theory governing the Rubik's cube is David Joyner's *Adventures in Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys* [28]. Readers who are particularly interested in permutation groups are, of course, referred to the standard works Cameron [7] and Dixon-Mortimer [17].
A far less mathematical but perhaps equally interesting resource is *Rubik's Cubic Compendium* [56] by Ernő Rubik, Tamás Varga, Gerzson Kéri, György Marx, and Tamás Vekerdy, which covers a wide variety of topics regarding the Rubik's cube.

Chapter 3

Difficult computational problems

The Basic Mathematical Problem is to restore the cube from any random pattern back to the original pattern in which each face is of a single colour. Masochists with a mathematical background may wish to start solving this problem at this point.

— David Singmaster, *Notes on Rubik's Magic Cube*

Any cryptographic algorithm first requires the definition of a problem that is infeasible to solve for an attacker. The security of the cryptographic scheme is then based on this infeasibility. Therefore, this chapter briefly discusses difficult computational problems related to Rubik's cubes.

3.1 The conjugacy problem

One of the most important decision problems in group theory is based on the conjugacy group action.

Definition 3.1 (Conjugacy group action)**.** Let G be a group. The *conjugation* on G is the group action $x^y := y^{-1}xy$ for $x, y \in G$.

We will also use the notation x^Y for $x \in G$ and $Y \subseteq G$ to denote the set $x^Y := \{x^y \mid y \in Y\}$.

Based on the definition of the conjugacy group action, we define the symmetric conjugacy relation for elements of groups.

Definition 3.2 (Conjugacy relation)**.** Let G be a group and $x, y \in G$. We say that x and y are *conjugate* if and only if there is a $z \in G$ such that $x^z = y$.

The conjugacy decision problem was first defined in 1911, more than 60 years before the invention of the Rubik's cube [12].

Definition 3.3 (Conjugacy decision problem)**.** Given a group G and two elements $x, y \in G$, determine whether x and y are conjugate.

The conjugacy decision problem is known to be undecidable for many infinite groups [42, 43], but if G is finite, then the problem is, of course, decidable. In fact, for many finite and infinite groups, the problem is trivial.

Example. Let A be an abelian group and $x, z \in A$. Because A is abelian, the group operation is commutative, thus $x^z = z^{-1}zx = x$. Therefore, x is conjugate only to itself.

However, for non-abelian permutation groups, no polynomial-time decision algorithms are known for the conjugacy decision problem.

Example. The smallest non-abelian symmetric group is S_3. Determine the conjugacy class of $\sigma = (1\ 2\ 3)$, that is, $\{\sigma^y \mid y \in S_3\}$.

$$\begin{array}{ccc} \sigma^{(1)\,(2)\,(3)} = (1\ 2\ 3) & \sigma^{(1\ 2)\,(3)} = (1\ 3\ 2) & \sigma^{(1\ 3)\,(2)} = (1\ 3\ 2) \\ \sigma^{(1)\,(2\ 3)} = (1\ 3\ 2) & \sigma^{(1\ 2\ 3)} = (1\ 2\ 3) & \sigma^{(1\ 3\ 2)} = (1\ 2\ 3) \end{array}$$

Therefore, the conjugacy class of σ is $\{\,(1\ 2\ 3), (1\ 3\ 2)\,\} \subset S_3$.

Interestingly, the well-known graph isomorphism problem is reducible to the conjugacy decision problem for permutation groups in polynomial time [59]. Both problems are in NP, but neither has known polynomial-time decision algorithms, therefore, it is unknown if they are in P.

If $\mathsf{P} = \mathsf{NP}$, then both the conjugacy decision problem for permutation groups and the graph isomorphism problem are in P. Even if $\mathsf{P} \neq \mathsf{NP}$, both problems might still be in P despite no such polynomial-time algorithms being known [59]. However, if these problems are not in P, it also seems unlikely that they could be NP-complete due to the consequences for the polynomial hierarchy, which would collapse if graph isomorphism was NP-complete [58, 59]. This result leads to a third complexity class that these problems could be in.

Definition 3.4 (NP-intermediate)**.** NPI is the subset of $\mathsf{NP} \setminus \mathsf{P}$ that only contains problem that are not NP-complete.

If $\mathsf{P} = \mathsf{NP}$, then, of course, $\mathsf{NPI} = \emptyset$. In 1975, Richard E. Ladner proved that the inverse is also true [36]:

Theorem 3.5 (Ladner's theorem)**.** *If* $\mathsf{P} \neq \mathsf{NP}$, *then* $\mathsf{NPI} \neq \emptyset$.

Proof. See [36]. □

If $\mathsf{P} \neq \mathsf{NP}$, the conjugacy decision problem for permutation groups and the graph isomorphism problem could be in NPI. Another candidate for NPI is the integer factorization problem, which the commonly used Rivest–Shamir–Adleman public-key cryptosystem (RSA) is based on [52]. It was later shown that the integer factorization problem is in BQP [62], which means that quantum computers can perform integer factorization in polynomial time and with small error probability, rendering the security provided by RSA insufficient against quantum computing attacks. However, it is unknown if other NPI candidates, including the graph isomorphism problem and the conjugacy decision problem for permutation groups, are in BQP as well.

Regardless of whether the conjugacy decision problem for permutation groups is in P, in NPI, or even NP-complete, it might be in P for certain classes of permutation groups [59]. In particular, it might be possible to solve the conjugacy decision problem for the class of permutation groups representing Rubik's cubes. However, again, no such polynomial-time algorithms are known.

Recall that an element of a Rubik's cube group can be seen either as a state of the cube or as a change of the state of the cube (i.e., a sequence of turns). Therefore, in terms of a Rubik's cube (of arbitrary size n), the conjugacy decision problem can be interpreted as follows: given a Rubik's cube state $y \in \mathfrak{R}_n$ and a sequence of turns $x \in \mathfrak{R}_n$, decide if there is a sequence of turns $z \in \mathfrak{R}_n$, such that, beginning with the solved state of the Rubik's cube, applying z^{-1} followed by x followed by z results in y.
In the context of cryptography, the corresponding search problem is often more useful than the decision problem itself.

Definition 3.6 (Conjugacy search problem)**.** Given a group G and two elements $x, y \in G$, find $z \in G$, such that $x^z = y$.

A variety of cryptosystems have been proposed based on the conjugacy problem over different groups, such as the Anshel-Anshel-Goldfeld key exchange scheme [1] (see Section 6.2). This problem will also play a central role in Chapter 5.

3.2 Solving the cube in a fixed number of turns

The process of restoring a Rubik's cube to its solved state through a sequence of allowed turns is the primary aspect of the puzzle. As such, it has been thoroughly researched, and a number of different methods and algorithms exist for solving the Rubik's cube. Despite the complexity of the puzzle, there are straightforward methods that reliably solve the Rubik's cube. Such algorithms usually consist of different phases that gradually restore the cube to a solved state [56, p. 98].
When solving a Rubik's cube by hand, these methods are important due to the complexity of the puzzle, and the time it takes a person to solve the cube depends more on the chosen method and on the speed of each physical movement than on the number of turns. For example, the current world record for a single solve of the 3x3x3 Rubik's cube is 3.47 seconds, during which Yusheng Du performed 27 turns (in the half turn metric) [72].

For computers, however, the time to physically solve a Rubik's cube is not necessarily meaningful, and finding a sequence of moves that solves the Rubik's cube is not particularly challenging.

Remark 3.7. For the definitions of the following computational problems, we assume that the Rubik's cube is only solved when all facelets are in their original positions, i.e., we assume that only the identity permutation represents a solved state. As described in Section 2.4, in the traditional interpretation of the puzzle, this is only true for the standard 3x3x3 Rubik's cube. Of course, the following definitions can be adapted to better represent the traditional interpretation, however, there is little value in doing so as part of this thesis. Allowing only the identity permutation as the single solved state is logical in the mathematical model and simplifies problem definitions and connections to known computational problems. Conversely, one can write numbers onto the facelets of any Rubik's cube and attempt to not only solve it but also to restore all facelets to their original positions, thus obtaining the problem as described here.

With this in mind, formally, solving the puzzle is the following search problem.

Definition 3.8 (Solving a Rubik's cube of size n). Given $\sigma \in \mathfrak{R}_n$ and a turn metric $T = \{ t_1, \ldots, t_m \}$, find $r \in \mathbb{N}_0$ and a sequence $i_1, \ldots, i_r$, such that $\sigma t_{i_1} t_{i_2} \ldots t_{i_r} = \mathrm{id}_{\mathfrak{R}_n}$.

For any state $\sigma \in \mathfrak{R}_n$, an infinite number of solutions exist. Even when $\sigma = \mathrm{id}_{\mathfrak{R}_n}$, any sequence that first scrambles and then unscrambles the Rubik's cube is a solution.
Upon closer inspection, the aforementioned search problem is remarkably similar to a better known computational problem that was listed along with the conjugacy decision problem (see Definition 3.3) by Dehn in 1911 [12].

Definition 3.9 (Generalized word problem for groups).
Let G be a group and let $g, g_1, \ldots, g_m \in G$. Let $H = \langle g_1, \ldots, g_m \rangle$. Decide if $g \in H$ by finding $i_1, \ldots, i_r$ such that $g = g_{i_1} g_{i_2} \ldots g_{i_r}$ if such a sequence exists.

Of course, the generalized word problem is decidable for finite groups. However, like the conjugacy decision problem (see Definition 3.3), it is undecidable in general [42, 43].

A common abstraction of the generalized word problem for permutation groups can be achieved by constructing a Cayley graph for the group [8]. The problem is then equivalent to finding a path from the identity permutation to the given permutation within the constructed graph.

Theorem 3.10. *The problem of solving a Rubik's cube of size n is equivalent to the generalized word problem for groups with $G = H = \mathfrak{R}_n$.*

Proof. Let $T = \{ t_1, \dots, t_m \}$ be defined as in Definition 3.8. Let $g_i := t_i^{-1}$ for $1 \leq i \leq m$. We have $g_i = t_i^3$ and $t_i = g_i^3$ and, therefore, $\mathfrak{R}_n = \langle t_1, \dots, t_m \rangle = \langle t_1^{-1}, \dots, t_m^{-1} \rangle = \langle g_1, \dots, g_m \rangle$. A solution to the generalized word problem is of the form $g = g_{i_1} g_{i_2} \dots g_{i_r}$, which is equivalent to $g g_{i_r}^{-1} \dots g_{i_2}^{-1} g_{i_1}^{-1} = \mathrm{id}_{\mathfrak{R}_n}$, which is equivalent to $g t_{i_r} \dots t_{i_2} t_{i_1} = \mathrm{id}_{\mathfrak{R}_n}$, which means that $i_r, \dots, i_2, i_1$ is a solution for the Rubik's cube. □

Remark 3.11. The above theorem ignores the decision aspect of the generalized word problem for groups, i.e., any element of H is an element of G because $G = H$. A closer approximation might be the generalized word problem with $G = S_{\mathrm{dom}(\mathfrak{R}_n)}$ and $H = \mathfrak{R}_n$. Semantically, this problem arises when given a Rubik's cube whose non-center facelets may have been reordered arbitrarily (e.g., by peeling off the stickers before gluing them back on). A solution exists if and only if the same pattern could have been achieved through a sequence of allowed turns. However, the decision problem is easily solved by attempting to solve the search problem using one of the aforementioned methods.

The problem becomes significantly more difficult when the number of turns in the solution sequence is limited by some non-trivial upper bound. The modified form of the generalized word problem for groups (see Definition 3.9) that additionally restricts the length r of the sequence is often referred to as the *factorization problem* for groups [49], which Lubotzky classified as a "noncommutative analog of the discrete logarithm problem" [41, p. 102]. Rephrasing the problem in terms of the Rubik's cube, we arrive at the following definition.

Definition 3.12 (Solving a Rubik's cube of size n with a limited number of turns)**.** Given $\sigma \in \mathfrak{R}_n$, a turn metric $T = \{ t_1, \dots, t_m \}$, and some $\ell > 0$, find a sequence $i_1, \dots, i_r$ for some $0 \leq r \leq \ell$, such that $\sigma t_{i_1} t_{i_2} \dots t_{i_r} = \mathrm{id}_{\mathfrak{R}_n}$.

Unlike the problem of solving a Rubik's cube in any number of turns (see Definition 3.8), this problem might not have a solution.

Example. The state F^2 can be solved in no more than one turn in HTM, STM, and OBTM, but not in QTM, QSTM, and CQLTM.

Definition 3.13 (Solving a Rubik's cube of size n with a fixed number of turns)**.** Given $\sigma \in \mathfrak{R}_n$, a turn metric $T = \{ t_1, \dots, t_m \}$, and some $\ell > 0$, find a sequence $i_1, \dots, i_\ell$, such that $\sigma t_{i_1} t_{i_2} \dots t_{i_\ell} = \mathrm{id}_{\mathfrak{R}_n}$.

A solution in no more than ℓ moves does not necessarily imply the existence of a solution in exactly ℓ moves.

Example. The state F can be solved in no more than four turns in QTM, but it cannot be solved in precisely four turns in the same turn metric.

An exhaustive ("brute force") search for a solution to the problem of solving a Rubik's cube in a fixed number of turns takes time $\mathcal{O}\left(|T|^\ell\right)$. However, it is possible to reduce the search space at the cost of increasing the space complexity.

Theorem 3.14. *A solution can be found in time $\mathcal{O}\left(2 \cdot |T|^{\ell/2}\right)$ if ℓ is even.*

Proof. The problem is approached from the left and right side separately. In cryptography, this is referred to as a meet-in-the-middle attack. The equation $\sigma t_{i_1} t_{i_2} \dots t_{i_\ell} = \mathrm{id}_{\mathfrak{R}_n}$ (see Definition 3.13) is equivalent to

$$\sigma t_{i_1} t_{i_2} \dots t_{i_{\ell/2}} = (t_{i_\ell})^{-1} \dots (t_{i_{\ell/2+1}})^{-1}. \tag{3.1}$$

Compute all left sides of Equation 3.1. Then, while computing all right sides of Equation 3.1, check for equality with any of the previously computed left sides. This method runs in $\mathcal{O}\left(2 \cdot |T|^{\ell/2}\right)$. □

Perhaps the most interesting variation of the problem is the search problem seeking an optimal solution to the Rubik's cube.

Definition 3.15 (Solving a Rubik's cube of size n optimally)**.** Given $\sigma \in \mathfrak{R}_n$ and a turn metric $T = \{ t_1, \dots, t_m \}$, find a sequence $i_1, \dots, i_r$ for some $r \geq 0$, such that

- $\sigma t_{i_1} t_{i_2} \dots t_{i_r} = \mathrm{id}_{\mathfrak{R}_n}$, and
- $\sigma t_{i'_1} \dots t_{i'_{r'}} \neq \mathrm{id}_{\mathfrak{R}_n}$ for all $0 \leq r' < r$ and all sequences $i'_1, \dots, i'_{r'}$.

An optimal solution always exists, and it seems reasonable to assume that an optimal solution will be unique or almost unique [70].

As mentioned above, solving the generalized word problem for a permutation group is equivalent to finding a path in the Cayley graph representing the group [8]. Finding an optimal solution as in Definition 3.15 is, therefore, equivalent to finding a shortest path in the Cayley graph of the Rubik's cube group. Unfortunately, the Cayley graph itself consists of $|\mathfrak{R}_n|$ vertices, and each vertex has $|T|$ outgoing edges. The graph is unweighted and as such has no distance measure other than the length of the shortest path from each vertex to the target node. The shortest path algorithm for this class of graphs must not traverse more than a logarithmic number of vertices to achieve polynomial time because the number of vertices grows exponentially in n [14].

It is unknown if the problem is NP-complete [30, p. 27]. In a characterization of the class NP, Stephen A. Cook used Rubik's cubes as an example of a problem that is difficult to solve but whose solution is simple to verify, which is the very nature of many problems in NP [10]. Demaine et al. designed a puzzle based on the concept of Rubik's cubes and demonstrated that optimally solving it is NP-hard, but the result does not apply to Rubik's cubes [14]. Volte et al. also relate the problem of solving a Rubik's cube in a limited number of turns to problems that are provably hard [70, p. 89].

The difficulty of these problems is apparent both in manual solving methods and in software implementations of Rubik's cube solvers. Manual methods rarely even get close to 20 HTM turns for the 3x3x3 Rubik's cube [56, 66] even though an optimal solution would require 20 HTM turns at most (see Section 2.4).

Most current software implementations do not solve the same Rubik's cube optimally either. The most commonly used computer algorithm is Kociemba's two-phase algorithm, which solves 3x3x3 Rubik's cubes in less than 20 HTM moves on average. Requiring only a fraction of a second to solve a Rubik's cube on modern hardware, it is the fastest known algorithm that produces near-optimal solutions [55]. Kociemba also developed an optimal solver for 3x3x3 Rubik's cubes. Even on modern hardware, it can take significant amounts of time for the algorithm to solve a single position [35].

While the complexity of the 3x3x3 Rubik's cube is astonishing, it is still feasible to find near-optimal and even optimal solutions with existing computational resources in short amounts of time. However, as n grows, the complexity of the problem grows.

The order of the group $\mathfrak{R}_3$ is only 4.3×10^{19} and it still took many years for its diameter to be determined in the half turn metric, and only through complicated reductions and years of CPU time. Other properties, including the group's diameter in some other turn metrics, are still unknown [55].

Finding optimal or near-optimal solutions in the larger group $\mathfrak{R}_5$, whose order is 2.6×10^{90}, appears to be significantly more challenging. Both manual methods and state-of-the-art computer algorithms, such as Daniel Walton's solver for arbitrarily large Rubik's cubes [71], typically reduce the problem of solving larger cubes to the problem of solving a 3x3x3 cube through a sequence of turns. Therefore, solutions for larger cubes usually consist of a sequence reducing the problem to that of solving a smaller Rubik's cube followed by a sequence of turns solving the smaller cube.

As mentioned above, it seems likely that an optimal solution would be unique or almost unique even for larger Rubik's cubes. As a consequence of this assumption, it is highly unlikely that such a solution consisting of two distinct phases is optimal or near-optimal.

Lastly, note that Demaine et al. proved that God's number (see Definition 2.13) grows with the Rubik's cube size n as $\Theta\left(n^2/\log n\right)$ and that the group order is in $2^{\Theta\left(n^2\right)}$ [14]. For the 5x5x5 Rubik's cube, the exact number is unknown (in any turn metric), but one could find a lower bound using the technique in Section 2.4 and an upper bound by considering a specific reduction method from the problem of solving the 5x5x5 Rubik's cube to the problem of solving the 3x3x3 Rubik's cube. The upper bound is the maximum number of turns required by the reduction method plus the maximum number of turns required to solve the 3x3x3 Rubik's cube (e.g., 20 in HTM). However, with current reduction methods, the difference between thereby derived upper and lower bounds is too big to draw meaningful conclusions.

It is very likely that, regardless of the size of a Rubik's cube, an optimal solution would not consist of distinct phases. Ernő Rubik himself wrote: "Increasing the number of elements may be interesting, but in my opinion, it does not really add anything new." He added that the 3x3x3 cube "gives the maximum amount of information with the minimum number of elements" [56, p. 15], referring to the fact that larger cubes still only consist of the same structural elements, which are corner cubies, edge cubies, and center cubies. This supports the assumption that solving larger cubes optimally is unlikely to be achieved through processes that use reduction to smaller cubes.

Given the lack of known polynomial-time solutions to the generalized word problem for permutation groups (see Definition 3.9), the proven increase of the groups' diameters and orders as the size of the Rubik's cube increases, and the lack of known polynomial-time optimal solvers, it does indeed seem appropriate to base cryptographic schemes on the assumed difficulty of the problems from Definition 3.13 and Definition 3.15.

3.3 Suggested literature

While Dehn's *Über unendliche diskontinuierliche Gruppen* [12] must be mentioned here, more recent works such as those written by Charles F. Miller III [43, 42] are far more detailed and benefit from additional decades of research since Dehn's work.

There are probably very few people who have spent more time studying the Rubik's cube group's diameter and God's number than Rokicki et al. [55]. Their effort to answer a question that had been open for decades goes far beyond the mere computational power that was required for the computer-assisted proof. Similarly, Demaine et al. [14] provide some of the most recent results concerning the (asymptotic) complexity of Rubik's cubes and possible relations to NP-completeness. A survey of NP-complete puzzles can also be found in [30].

Chapter 4

A zero-knowledge protocol

It is possible to prove that some theorems are true without giving the slightest hint of why this is so.
— Blum et al., *Non-Interactive Zero-Knowledge and Its Applications*

Volte et al. proposed a zero-knowledge protocol based on Rubik's cubes in 2013, which they also generalized to other non-abelian groups [70]. The concept of zero-knowledge proofs was introduced by Goldwasser, Micali, and Rackoff [23, 24], who were awarded the very first *Gödel prize* for this contribution. Later, Goldreich et al. described zero-knowledge proofs as "proofs that yield nothing but their validity" [22]. In the context of cryptography, zero-knowledge protocols define interactions between a prover and a verifier, during which the prover has to prove a previously made claim to the verifier without providing information that would allow the verifier to prove the same (or any) claim themselves. In this case, the public key of the prover is a seemingly random state of a Rubik's cube, and the secret knowledge of the prover is a sequence of d moves that solves the Rubik's cube. The purpose of the zero-knowledge protocol is for the prover to prove their ability to solve the publicly known state of the Rubik's cube in d moves without disclosing any information that would allow the verifier to solve the Rubik's cube in the same number of moves themselves.

Assuming that only the prover knows the correct sequence of moves, such a zero-knowledge protocol can be used by the verifier to confirm the identity of the prover that is associated with the prover's public key. The challenge is, of course, for the prover to prove that they know the sequence of moves, without actually disclosing any part of it.

Thanks to Goldreich et al., it is well known that zero-knowledge proofs can be constructed for all languages in NP, however, the general construction requires a reduction of the language to 3-Colorability [22, p. 722] and does not necessarily lead to efficient protocols. Thus, even when the existence of a zero-knowledge protocol for a language is already known, it still makes sense to search for a protocol that is specific to the language and more efficient than the general construction by design [70].

4.1 Repositioning groups

The zero-knowledge protocol makes extensive use of repositioning groups.

Definition 4.1 (Repositioning group)**.** Let G be a group, let $\mathcal{F} = \{ f_1, \ldots, f_\alpha \} \subset G$, and let $H < G$. We say that H is a *repositioning group* of $\mathcal{F}$ if $f_1^H = \mathcal{F}$.

Of course, for any $\mathcal{F} \subset G$, a repositioning group can only exist if the elements of $\mathcal{F}$ are conjugate (see Definition 3.2).

Theorem 4.2. *Let H be a repositioning group of $\mathcal{F}$. If $\tau \in_R H$ is drawn from a discrete uniform probability distribution, then $f^\tau \in \mathcal{F}$ is a random variable with uniform probability distribution for all $f \in \mathcal{F}$.*

Proof. By contradiction. Let $i, j, k, \ell \in \{ 1, 2, \ldots, \alpha \}$ be arbitrary and, w.l.o.g., assume that $\mathcal{P}(f_i^\tau = f_j) > \mathcal{P}(f_k^\tau = f_\ell)$ for $\tau \in_R H$. Because τ is selected from a discrete uniform probability distribution, this implies that there are more $\tau \in H$ such that $f_i^\tau = f_j$ than there are $\tau' \in H$ with $f_k^{\tau'} = f_\ell$.

Let $\tau_i, \tau_j, \tau_k, \tau_\ell \in H$ be some elements of H such that $f_1^{\tau_i} = f_i$, $f_1^{\tau_j} = f_j$, $f_1^{\tau_k} = f_k$, and $f_1^{\tau_\ell} = f_\ell$. For any $\tau \in H$, let $\tau' = \tau_k^{-1}\tau_i\tau\tau_j^{-1}\tau_\ell$. Then

$$f_i^\tau = f_j \quad \Longrightarrow \quad f_k^{\tau'} = f_k^{\tau_k^{-1}\tau_i\tau\tau_j^{-1}\tau_\ell} = f_i^{\tau\tau_j^{-1}\tau_\ell} = f_j^{\tau_j^{-1}\tau_\ell} = f_1^{\tau_\ell} = f_\ell.$$

Therefore, for every $\tau \in H$ with $f_i^\tau = f_j$, there is a $\tau' \in H$ with $f_k^{\tau'} = f_\ell$, which contradicts the assumption that there are more such values for τ than there are for τ'. Thus, for $\tau \in_R H$, we have $\mathcal{P}(f_i^\tau = f_j) = \mathcal{P}(f_k^\tau = f_\ell)$ for all i, j, k, ℓ, which means that f^τ is a random variable with uniform probability distribution for all $f \in \mathcal{F}$. □

A repositioning group H masks elements of $\mathcal{F}$. If $f_i^\tau = f_j$ for some $f_i \in \mathcal{F}$ and $\tau \in H$, then knowing f_j gives no information about f_i unless τ is disclosed. Yet, knowing only f_j and that τ is some element of H, anyone can check if $f_i \in \mathcal{F}$ since $f_i \in \mathcal{F} \iff f_j \in \mathcal{F}$.

4.2 Commitment schemes

The protocol also requires a commitment scheme. Informally, the prover needs the ability to commit to a statement without disclosing the statement itself. Later, the prover can reveal the statement along with a key that was used to produce the commitment, allowing the other party to verify the commitment.

Definition 4.3 (Commitment scheme)**.** A commitment scheme for a message space $\mathcal{M}$ is a key space K, a commitment space C, and a function $\mathrm{Com}\colon K \times \mathcal{M} \to C$, which is

- *statistically hiding*, meaning that the distributions $\mathrm{Com}(k, x)$ and $\mathrm{Com}(k', y)$ are computationally indistinguishable for all $(x, y) \in \mathcal{M}^2$ when $k, k' \in_R K$, and
- *computationally binding*, meaning that the probability of finding $k, k' \in K$ and $x, y \in \mathcal{M}$ with $(k, x) \neq (k', y)$ and $\mathrm{Com}(k, x) = \mathrm{Com}(k', y)$ in polynomial time is negligible.

In early work on commitment schemes, the two properties are often referred to as *secrecy* and *non-ambiguity* instead of *hiding* and *binding* [26, p. 209-210].

Commitment schemes typically operate in two phases, which are *commit* and *reveal*. When the verifier does not know the key $k \in K$, the commitment scheme ensures that the verifier cannot determine what message $m \in \mathcal{M}$ the prover committed to based on $c \in C$ because the commitment function is statistically hiding. On the other hand, when the protocol requires the prover to disclose $(k, m) \in K \times \mathcal{M}$ for a previously made commitment $c = \mathrm{Com}(k, m)$, finding a different $(k', m') \in K \times \mathcal{M}$ with $\mathrm{Com}(k', m') = c$ is intractable for the prover. Thus, in this case, the commitment scheme ensures that the prover has to truthfully disclose (k, m) during the *reveal* phase.

Volte et al. do not require the use of any specific commitment scheme [70]. Message authentication functions, such as the *Keyed-Hash Message Authentication Code* (HMAC) [47], are believed to be both statistically hiding and computationally binding.

4.3 Protocol

Following the presentation by Volte et al. [70], this section describes the general protocol for groups that have a repositioning group. The parameters of the protocol are

- a group G,
- a set $\mathcal{F} = \{ f_1, f_2, \dots, f_\alpha \} \subset G$,
- a repositioning group $H \leq G$ of $\mathcal{F}$ (see Definition 4.1), and
- a number of moves $d \geq 3$.

These parameters implicitly define the subgroups $G_R \coloneqq \langle \mathcal{F} \rangle$ and $G' \coloneqq \langle \mathcal{F}, H \rangle$.

Additionally, let $\mathrm{Com}\colon K \times G \to C$ be a commitment function, where K is the key space and C is the commitment space (see Definition 4.3).

The prover's secret key is a sequence $(i_1, i_2, \dots, i_d) \in \{ 1, 2, \dots, \alpha \}^d$, where α is the number of generators. The public key x_0 is the element of G_R such that $x_0 f_{i_1} f_{i_2} \dots f_{i_d} = \mathrm{id}_{G_R}$. Equivalently, $x_0 \coloneqq (f_{i_1} f_{i_2} \dots f_{i_d})^{-1}$.

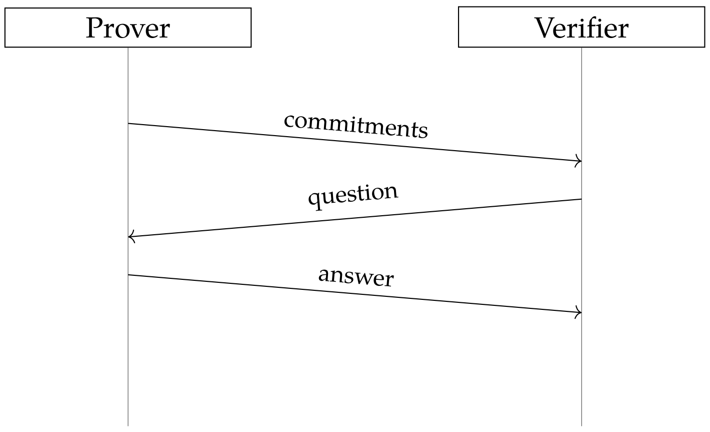


Figure 4.1: Typical structure of a single round in an interactive zero-knowledge protocol

The key generation algorithm is shown in Algorithm 4.1. In terms of a Rubik's cube, x_0 is a state of the cube whose solution is the turn sequence $f_{i_1} f_{i_2} \dots f_{i_d}$.

Algorithm 4.1 Key generation

```
Output: secret key $(i_1, i_2, \dots, i_d)$, public key $x_0$
1: $\sigma \leftarrow \mathrm{id}_{G_R}$
2: for $j \leftarrow 1$ to $d$ do
3:     $i_j \leftarrow$ random integer from $\{ 1, 2, \dots, \alpha \}$
4:     $\sigma \leftarrow \sigma f_{i_j}$                ▷ $f_{i_j}$ is an element of $\mathcal{F}$
5: $x_0 \leftarrow \sigma^{-1}$
```

As usual, the zero-knowledge protocol defines an interaction as a single round during which the prover provides a set of commitments, followed by the verifier asking a question and the prover answering the question (see Figure 4.1). The verifier uses the answer to verify a subset of the commitments provided in the first step. A truthful prover will always provide a valid answer.

Algorithm 4.2 Prover: round initialization

```
Input: secret key $(i_1, \ldots, i_d)$
Output: secret state $(\tau, \sigma_0, \ldots, \sigma_d, k_*, k_0, \ldots, k_d)$, commitments $c_0, s_0, \ldots, s_d$
1: $\tau \leftarrow$ random element of $H$
2: $\sigma_0 \leftarrow$ random element of $G' = \langle \mathcal{F}, H \rangle$
3: for $j \leftarrow 1$ to $d$ do
4:     $\sigma_j \leftarrow (f_{i_j}^{\tau})^{-1} \sigma_{j-1}$
5: $(k_*, k_0, \ldots, k_d) \leftarrow$ random element of $K^{d+2}$
6: $c_0 \leftarrow \mathrm{Com}(k_*, \tau)$
7: for $i \leftarrow 0$ to $d$ do
8:     $s_i \leftarrow \mathrm{Com}(k_i, \sigma_i)$
```

At the beginning of each round (see Algorithm 4.2), the prover picks $\tau \in_R H, \sigma_0 \in_R G', k_*, k_0, \ldots, k_d \in_R K$ at random and computes $\sigma_j := (f_{i_j}^{\tau})^{-1} \sigma_{j-1}$ for $1 \leq j \leq d$. The prover then provides commitments c_0 and $s_0, \ldots, s_d$ for τ and $\sigma_0, \ldots, \sigma_d$, respectively. The keys that are used for these commitments are k_* and $k_0, \ldots, k_d$, respectively.

Algorithm 4.3 Prover: generate answer

```
Input: secret key $(i_1, \ldots, i_d)$, state $(\tau, \sigma_0, \ldots, \sigma_d, k_*, k_0, \ldots, k_d)$, question $q$
Output: answer $a \in (H \times G' \times K^3) \cup (\mathcal{F} \times G' \times K^2)$
1: if not the first invocation of this algorithm in this round then
2:     abort                ▷ Prevent cheating verifiers from extracting information.
3: if $q = 0$ then
4:     $a \leftarrow (\tau, \sigma_0, k_*, k_0, k_d)$
5: else
6:     $a \leftarrow (f_{i_q}^{\tau}, \sigma_q, k_{q-1}, k_q)$
```

The verifier picks $q \in_R \{0, \ldots, d\}$. This is the question. If $q = 0$, the answer provided by the prover (see Algorithm 4.3) consists of τ, σ_0, k_*, k_0, and k_d. Otherwise, the answer consists of $f_{i_q}^{\tau}$, σ_q, k_{q-1}, and k_q.

Algorithm 4.4 Verifier: verify answer

```
Input: public key $x_0$, commitments $c_0, s_0, \ldots, s_d$, question $q$, answer $a$
 1: if $q = 0$ then
 2:     $(\tau, \sigma_0, k_*, k_0, k_d) \leftarrow a$
 3:     if $\tau \notin H$ then
 4:         reject
 5:     $\sigma_d \leftarrow x_0^\tau \sigma_0$
 6:     if $\mathrm{Com}(k_*, \tau) \neq c_0$ or $\mathrm{Com}(k_0, \sigma_0) \neq s_0$ or $\mathrm{Com}(k_d, \sigma_d) \neq s_d$ then
 7:         reject
 8: else
 9:     $(f_{i_q}^\tau, \sigma_q, k_{q-1}, k_q) \leftarrow a$
10:     if $f_{i_q}^\tau \notin \mathcal{F}$ then
11:         reject
12:     $\sigma_{q-1} \leftarrow f_{i_q}^\tau \sigma_q$
13:     if $\mathrm{Com}(k_{q-1}, \sigma_{q-1}) \neq s_{q-1}$ or $\mathrm{Com}(k_q, \sigma_q) \neq s_q$ then
14:         reject
```

The verifier checks commitments based on the received answer (Algorithm 4.4).

(i) If the question was $q = 0$, the verifier computes $\sigma_d = x_0^\tau \sigma_0$. They can then verify that $\tau \in H$ (line 3) and that the commitments for τ, σ_0, and σ_d are valid (line 6). Intuitively, this is equivalent to verifying that the prover committed to a sequence of d moves that indeed transforms x_0 into id_{G_R} (i.e., solve the Rubik's cube), but the verifier has no way of knowing the moves that were used to do so. In particular, the verifier cannot even tell if all of the moves are valid moves.

(ii) If the question was $q \neq 0$, the verifier computes $\sigma_{q-1} = f_{i_q}^\tau \sigma_q$. They can then verify that $f_{i_q}^\tau \in \mathcal{F}$ (without knowing i_q or τ) and that the commitments for σ_{q-1} and σ_q are valid. The verifier thus establishes that the q-th move in the sequence of d moves is indeed a valid move, but, because the verifier does not know τ, cannot obtain f_{i_q} from $f_{i_q}^\tau$ (see Theorem 4.2). Also, the verifier can neither tell if any of the other $d-1$ moves are valid, nor if the whole sequence of moves transforms x_0 into id_{G_R}.

Regardless of the question q, that a *cheating prover* could fool the verifier because the verifier only checks a subset of all commitments and conditions. However, as we will see in the next section, the probability that a cheating prover successfully fools a verifier is negligible when multiple rounds of the protocol are performed.

4.4 Zero-knowledge property of the protocol

The concept of interactive zero-knowledge proof systems extends the notion of interactive proof systems. An interactive proof system represents prover and verifier as an interactive pair of Turing machines and requires the protocol to be complete and sound. For a formal definition, readers are referred to Goldwasser et al. [23, p. 293]. Informally, completeness means an honest prover's ability to convince a verifier of the prover's knowledge, whereas soundness means that no cheating prover will be able to convince an honest verifier of a false statement with greater than negligible probability.

Remark 4.4. Note that the verifier is restricted to polynomial-time computation. Without this restriction, zero-knowledge protocols for languages in NP would be meaningless in the context of cryptography [24, p. 291].

Lemma 4.5. *The protocol is complete, that is, an honest prover will always be accepted by an honest verifier.*

Proof. An honest verifier will not deviate from the protocol as shown in Figure 4.1. During the protocol, only the verification algorithm (Algorithm 4.4) could reject the prover. We show that, if the prover is honest, the algorithm will not reject the prover.
If $q = 0$, the verifier computes $\sigma_d = x_0^\tau \sigma_0$ (in line 5 of Algorithm 4.4). This is correct because $\sigma_j = (f_{i_j}^\tau)^{-1}\sigma_{j-1}$ for $j \in \{1, \dots, d\}$ and

$$\begin{aligned}
\sigma_d &= (f_{i_d}^{\tau})^{-1}\sigma_{d-1} \\
&= (f_{i_d}^{\tau})^{-1}(f_{i_{d-1}}^{\tau})^{-1}\sigma_{d-2} \\
&= (f_{i_d}^{\tau})^{-1}(f_{i_{d-1}}^{\tau})^{-1}(f_{i_{d-2}}^{\tau})^{-1}\sigma_{d-3} \\
&\vdots \\
&= (f_{i_d}^{\tau})^{-1}(f_{i_{d-1}}^{\tau})^{-1}(f_{i_{d-2}}^{\tau})^{-1}\ldots(f_{i_1}^{\tau})^{-1}\sigma_0 \\
&= (\tau^{-1}f_{i_d}\tau)^{-1}(\tau^{-1}f_{i_{d-1}}\tau)^{-1}(\tau^{-1}f_{i_{d-2}}\tau)^{-1}\ldots(\tau^{-1}f_{i_1}\tau)^{-1}\sigma_0 \\
&= (\tau^{-1}f_{i_d}^{-1}\tau)(\tau^{-1}f_{i_{d-1}}^{-1}\tau)(\tau^{-1}f_{i_{d-2}}^{-1}\tau)\ldots(\tau^{-1}f_{i_1}^{-1}\tau)\sigma_0 \\
&= \tau^{-1}f_{i_d}^{-1}f_{i_{d-1}}^{-1}f_{i_{d-2}}^{-1}\ldots f_{i_1}^{-1}\tau\sigma_0 \\
&= \tau^{-1}(f_{i_1}\ldots f_{i_{d-2}}f_{i_{d-1}}f_{i_d})^{-1}\tau\sigma_0 \\
&= \tau^{-1}x_0\tau\sigma_0 \\
&= x_0^{\tau}\sigma_0.
\end{aligned}$$

If $q \neq 0$, the verifier computes $\sigma_{q-1} = f_{i_q}^{\tau}\sigma_q$ (in line 12 of Algorithm 4.4). This is correct because

$$\begin{aligned}
& & \sigma_q &= (f_{i_q}^{\tau})^{-1}\sigma_{q-1} \\
&\iff & f_{i_q}^{\tau}\sigma_q &= f_{i_q}^{\tau}(f_{i_q}^{\tau})^{-1}\sigma_{q-1} \\
&\iff & f_{i_q}^{\tau}\sigma_q &= \sigma_{q-1}.
\end{aligned}$$

Therefore, an honest prover will always pass all checks performed by an (honest) verifier and thus will not be rejected. $\square$

The protocol is also sound. For this property to hold, Goldwasser et al. require the probability of the verifier accepting a cheating prover to be negligible [23, p. 293]. In Lemma 4.6, we show that, during a single round of the protocol, the probability p that the verifier accepts a cheating prover is at most $\frac{d}{d+1}$. Then, after $r \in \mathbb{N}$ rounds, the probability is at most

$r \mapsto \left(\frac{d}{d+1}\right)^r$, thus negligible. We can also use this to find specific values for r in order to achieve desired upper bounds on the impersonation probability (see Section 4.4.1).

The remainder of this section roughly follows the presentation by Volte et al. [70] and adds some details and explanations.

Lemma 4.6. *During a single round, the probability of accepting a cheating prover is at most $\frac{d}{d+1}$.*

Proof. By contradiction. Assume that a cheating prover, i.e., a prover who does not know $(i_1, \ldots, i_d) \in \{1, \ldots, \alpha\}^d$ such that $x_0 f_{i_1} f_{i_2} \ldots f_{i_d} = \mathrm{id}_{G_R}$, can answer any question $q \in \{0, 1, \ldots, d\}$ correctly.

Recall that, because the commitment scheme is binding (see Definition 4.3), if there are multiple ways of obtaining a value from the prover, all of them must yield the same result, e.g., σ_0 obtained through $q = 0$ must have the same value as σ_0 obtained through $q = 1$.

Because the prover answers $q = 0$ correctly, $\mathrm{id}_{G_R} = x_0 \tau \sigma_0 \sigma_d^{-1} \tau^{-1}$ must hold. Additionally, because the prover answers $1 \le q \le d$ correctly, $\sigma_{q-1}\sigma_q^{-1} \in \mathcal{F}$ must hold. Let $u_j \in \{1, 2, \ldots, \alpha\}$ such that $f_{u_j} = \sigma_{j-1}\sigma_j^{-1}$ for $1 \le j \le d$. Combined, we have

$$\begin{aligned} x_0 f_{i_1} f_{i_2} \ldots f_{i_d} = \ \mathrm{id}_{G_R} &= x_0 \tau \sigma_0 \sigma_d^{-1} \tau^{-1} \\ &= x_0 \tau (\sigma_0 \sigma_1^{-1})(\sigma_1 \sigma_2^{-1}) \ldots (\sigma_{d-1}\sigma_d^{-1}) \tau^{-1} \\ &= x_0 \tau f_{u_1} f_{u_2} \ldots f_{u_d} \tau^{-1} \\ &= x_0 \tau f_{u_1} \tau^{-1} \tau f_{u_2} \tau^{-1} \ldots \tau f_{u_d} \tau^{-1} \\ &= x_0 f_{u_1}^{\tau^{-1}} f_{u_2}^{\tau^{-1}} \ldots f_{u_d}^{\tau^{-1}}. \end{aligned}$$

The prover knows $f_{u_i}^{\tau^{-1}} = \sigma_{i-1}\sigma^{-1}$. Let i_j be the index of $f_{u_j}^{\tau^{-1}}$ for $1 \le j \le d$. Clearly, this is a solution for the problem of finding such a sequence $(i_1, i_2, \ldots, i_d)$. This contradicts the assumption that the prover is a cheating prover, i.e., does not know a solution.

Therefore, a cheating prover cannot be able to answer all questions correctly. There must be at least one $q \in \{0, 1, \ldots, d\}$ for which the prover will answer incorrectly, thus the impersonation probability is at most $\frac{q}{q+1}$. □

Corollary 4.7. *The proposed protocol is an interactive proof system as defined by Goldwasser et al. [23, p. 293].*

Proof. This follows from Lemma 4.5 and Lemma 4.6, which have shown that the protocol has the required properties. □

For an interactive proof system to be an interactive zero-knowledge proof system, the proof must not yield anything other than its validity. For a formalization of this statement, readers are referred to Goldreich et al. [22, p. 696].

For example, digital signatures can also be used as means of authentication. The verifier chooses some random sequence a, and the prover chooses some random sequence b and provides a digital signature for a cryptographic hash value of (a, b). Since only an honest prover could provide a valid signature, this identifies the prover. The verifier cannot use the signature to impersonate the prover since another party would choose a different value for a, thus rendering the signature useless. However, this protocol is certainly not zero-knowledge. For example, the verifier can now use the signature to prove to a third party that they, at some point, interacted with the prover, because only the prover could have produced the signature.

This idea is reflected by definitions of zero-knowledge proofs that, for any polynomial-time verifier V, require the existence of a polynomial-time simulator M_V, whose set of possible outputs is indistinguishable from the set of transcripts of possible interactions between the (honest) prover and V [22, p. 696]. Intuitively, in an interactive zero-knowledge proof system, any transcript of an interaction between the prover and the verifier could have been created in the absence of the prover. This implies that neither the verifier nor anyone else reading the transcript of the interaction learns anything from the interaction.

Theorem 4.8. *The proposed interactive proof system is zero-knowledge.*

Proof. We show that, for any verifier V, a simulator M_V exists, which is accepted with probability $\frac{d}{d+1}$, and, when unsuccessful rounds are omitted from the transcript, the transcript is computationally indistinguishable from the transcript between V and an honest prover. The simulator M_V acts like an honest prover, except at the beginning of the round. The simulator chooses $q^* \in \{0, 1, \ldots, d\}$ by predicting that $q^* \neq q$, where $q \in \{0, 1, \ldots, d\}$ is the question that will be selected by the verifier V.

(i) If $q^* = 0$, the simulator expects to answer $q \in \{1, \ldots, d\}$, thus it only needs to ensure that $\sigma_{j-1}\sigma_j^{-1} \in \mathcal{F}$ for $1 \leq j \leq d$. Therefore, the simulator picks $\tau \in_R H$, $\sigma_0 \in_R G'$, and $f'_1, \ldots, f'_d \in_R \mathcal{F}$ randomly and computes $\sigma_j = (f'_j)^{-1}\sigma_{j-1}$ for $1 \leq j \leq d$.

(ii) Otherwise, if $q^* \neq 0$, the simulator expects to answer $q \in \{0, 1, \ldots, q^* - 1, q^* + 1, \ldots, d\}$, thus it must ensure that $\tau \in H$, $\sigma_d = \tau^{-1}x_0\tau\sigma_0$, and $\sigma_{j-1}\sigma_j^{-1} \in \mathcal{F}$ for $j \in \{1, \ldots, q^* - 1, q^* + 1, \ldots, d\}$. The simulator picks $\tau \in_R H$, $\sigma_0 \in_R G'$, and $f'_1, \ldots, f'_{q^*-1}, f'_{q^*+1}, \ldots, f'_d \in_R \mathcal{F}$. The simulator then computes

$$f'_{q^*} = (x_0 f'_1 \cdots f'_{q^*-1})^{-1}(f'_{q^*+1} \cdots f'_d)^{-1}$$

and $\sigma_j = (f'_j)^{\tau^{-1}}\sigma_{j-1}$ for $1 \leq j \leq d$. Note that f'_{q^*} is in G', but generally not in $\mathcal{F}$. (Otherwise, the sequence $f'_1, \ldots, f'_d$ would be a solution to the initial problem of finding such a sequence, which is presumed to be intractable.)

Only if the simulator's prediction was incorrect, i.e., if $q^* = q$, the simulated prover will be rejected by the verifier V. The probability of this is at most $\frac{1}{d+1}$.

The only aspect of the construction of M_V that depends on V is the prediction of $q \in \{0, 1, \ldots, d\}$. If V is an honest verifier and picks $q \in_R \{0, 1, \ldots, d\}$ randomly, then M_V picks $q^* \in_R \{0, 1, \ldots, d\}$, and omits the current round from the transcript whenever $q = q^*$. Otherwise, if V does not pick q randomly, then q is, by definition, predictable, and there is a simulator M_V that correctly predicts q. $\square$

4.4.1 Security and number of rounds

The security of the scheme is limited by the computational complexity of deriving the secret key $(i_1, \ldots, i_d) \in \{1, \ldots, \alpha\}^d$ from the public key $x_0 \in G_R$. According to Theorem 3.14, this is possible in $\mathcal{O}\left(2 \cdot \alpha^{d/2}\right)$. No better attack is known, therefore, the security is estimated at $2 \cdot \alpha^{d/2}$.

The impersonation probability during a single round is at most $\frac{d}{d+1}$. Because rounds are statistically independent, the impersonation probability after r rounds is at most $\left(\frac{d}{d+1}\right)^r$. Volte et al. approximate the number of rounds that are required in order for the impersonation probability to be at most 2^{-m} for some $m > 0$ as $r \approx md\ln(2)$ [70]. However, upon closer inspection, the exact number of rounds required to achieve a probability of 2^{-m} is

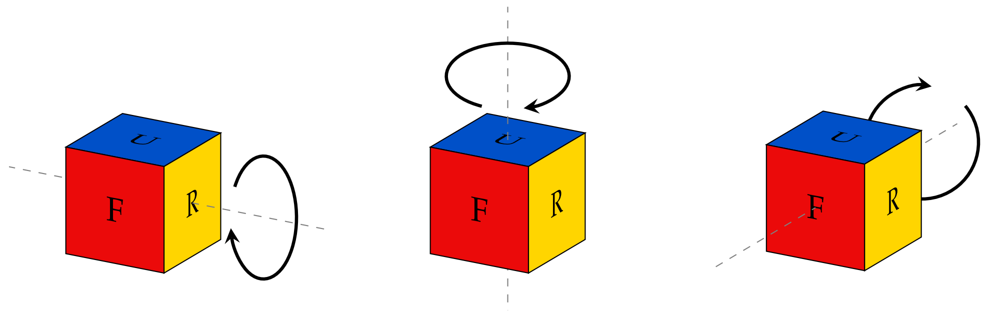
$$r = \left\lceil m/\log_2\left(\frac{d+1}{d}\right)\right\rceil .$$

4.5 Using the 3x3x3 Rubik's cube

The standard Rubik's cube's representation as a group is $\mathfrak{R}_3$. Conveniently, the group has an obvious repositioning group (see Definition 4.1) when using the standard set of generators. Recall that a repositioning group for a set of generators $\mathcal{F}$ is a permutation group H, such that each generator $f_i \in \mathcal{F}$ can be transformed into any other generator, i.e., H must fulfill

$$\{f_i^h \mid h \in H\} = \mathcal{F} \text{ for all } f_i \in \mathcal{F}.$$

The standard set of generators for $\mathfrak{R}_3$ is the set $\{U, L, F, R, B, D\}$, which each represent a clockwise quarter turn of the respective face of the cube. Recall from Section 2.1 that the faces are denoted by their relative orientation to the observer. This leads to a natural repositioning group, whose presentation here follows that provided by Volte et al. [70, p. 80] closely. By rotating the entire Rubik's cube in space, the orientation of the cube relative to the observer changes, which is equivalent to relabeling the faces without disrupting the



(a) Action h_1 (b) Action h_2 (c) Composition $h_1^{h_2}$

Figure 4.2: Visualization of the repositioning transformations h_1 and h_2

cube's structure. Each of the six generators is a turn of one of the six faces, and each face can be moved to the position of any other face by rotating the cube.

Each of the six faces can be rotated to the front and, regardless of which face is in the front position, the cube can be rotated around the axis that goes through the front face in 90 degree steps. Combined, there are $6 \times 4 = 24$ possible rotations. Therefore, the repositioning group H will consist of $|H| = 24$ permutations.

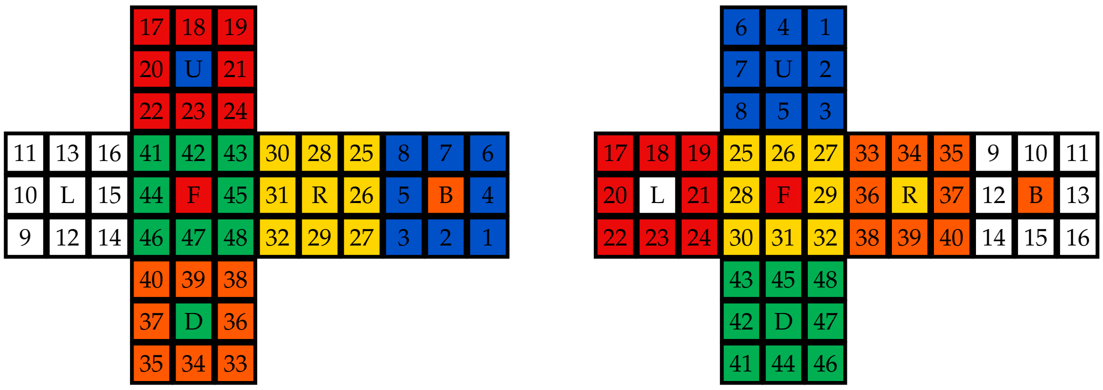
To generate H, only quarter rotations around two axes are required. Each can be expressed as the product of permutations rotating the three layers around the respective axis. The permutation h_1 rotates the cube around the axis going through the right (R) and left (L) faces (see Figure 4.2a), and h_2 rotates the cube around the axis going through the up (U) and down (D) faces (see Figure 4.2b).

$$h_1 := R\,\big((2\ 39\ 42\ 18)(7\ 34\ 47\ 23)\big)\,L^{-1}$$

$$h_2 := U\,\big((12\ 36\ 28\ 20)(13\ 37\ 29\ 21)\big)\,D^{-1}$$

Rotating the cube around the third axis is equivalent to $h_1^{h_2}$ (see Figure 4.2c). The repositioning group is $H := \langle h_1, h_2 \rangle$ and we have, indeed, $|H| = 24$. The other parameters are $G_R := \mathfrak{R}_3$ and $\mathcal{F} := \{\, U, L, F, R, B, D \,\}$.

Volte et al. suggest $d = 24$ [70, p. 80]. It was later shown that God's number in the quarter turn metric (see Definition 2.7) is 26 and, unlike the standard set of generators that is used



(a) Effect of h_1 (b) Effect of h_2

Figure 4.3: Effects of the repositioning transformations h_1 and h_2

as $\mathcal{F}$, the quarter turn metric allows counterclockwise turns [53]. Therefore, there must be Rubik's cube states that cannot be reached in only $d = 24$ turns from $\mathcal{F}$. However, at the same time, the order of $\mathfrak{R}_3$ is not much larger than $|\mathcal{F}|^{24}$, thus, increasing d would increase the probability of distinct secret keys representing the same public keys.

There are $|\mathcal{F}|^d = 6^{24} \approx 2^{62}$ secret keys, therefore, recovering the secret key from the public key is possible in about 2^{32} steps according to Theorem 3.14, and no better recovery attack is known [70].

The impersonation probability after r rounds is at most $\left(\frac{d}{d+1}\right)^r = \left(\frac{24}{25}\right)^r$. As discussed in Section 4.4.1, Volte et al. use an approximation to determine r such that the impersonation probability is at most 2^{-m}, which leads to $r = 500$ for $m = 30$ [70, p. 84]. However, as explained above, the exact number of rounds required to achieve a probability of 2^{-m} is $r = \left\lceil m / \log_2\left(\frac{d+1}{d}\right)\right\rceil$. In this case, $\left\lceil 30 / \log_2\left(\frac{25}{24}\right)\right\rceil = 510$ rounds are required to achieve the desired upper bound on the impersonation probability.

4.6 Using the 5x5x5 Rubik's cube

Unfortunately, the security provided by the zero-knowledge protocol using the 3x3x3 Rubik's cube that was presented in the previous section is insufficient in practice. However, larger Rubik's cubes appear to be much more difficult to solve (see Section 3.2). Therefore, Volte et al. also present parameters for the zero-knowledge protocol that are based on the 5x5x5 Rubik's cube [70, pp. 84-86].

The idea that was used to construct the repositioning group for the 3x3x3 Rubik's cube in Section 4.5, i.e., rotating the entire cube in space, does not work for larger Rubik's cubes because any turn metric, including the standard set of generators, consists of more than six generators. Because there are more generators than faces of the cube, rotating the cube is not sufficient for hiding a turn. For example, turns of the faces of the Rubik's cube (e.g., F_0) can still be distinguished from turns of the inner slices of the cube (e.g., F_1) even after rotating it in space.

Therefore, Volte et al. suggest the following construction [70]. Recall that $\mathfrak{R}_5 < S_{144}$. Now consider two 5x5x5 Rubik's cubes for a total of $2 \cdot 144 = 288$ facelets (excluding centers) and assign the facelet indices 1 to 144 to the left Rubik's cube and the indices 145 to 288 to the facelets of the right cube. The generators of $\mathfrak{R}_5$ now define the allowed moves (and thus all possible positions) of the left cube.

We define $\Delta\colon S_{144} \to S_{288}$ as $\Delta(\sigma) := \sigma'$, where $\sigma'(j) := \sigma(j - 144)$. Intuitively, given any permutation σ that operates on the left cube only, $\Delta(\sigma)$ is the permutation that represents the same permutation on the right cube only.

For any face $Q \in \{ U, L, F, R, B, D \}$ and $j \in \{ 0, 1 \}$, define $Q_j^{\sim} := Q_j \Delta(Q_{1-j})$. For example, $F_0^{\sim} = F_0 \Delta(F_1)$ is the permutation that simultaneously turns the front face of the left cube and the slice behind the front face of the right cube.

Instead of working in $\mathfrak{R}_5 = \langle \{ U_0, U_1, L_0, L_1, F_0, F_1, R_0, R_1, B_0, B_1, D_0, D_1 \} \rangle$, we use $\mathcal{F} := \{ U_0^{\sim}, U_1^{\sim}, L_0^{\sim}, L_1^{\sim}, F_0^{\sim}, F_1^{\sim}, R_0^{\sim}, R_1^{\sim}, B_0^{\sim}, B_1^{\sim}, D_0^{\sim}, D_1^{\sim} \}$, and thus have $G_R := \langle \mathcal{F} \rangle < S_{288}$.

Calculation shows that $|\mathfrak{R}_5| \approx 1.3 \times 2^{300}$, and $|G_R| \approx 1.5 \times 2^{364}$. Intuitively, the increase in the order of the group means that, for each possible state of the left cube, the right cube can be in one of many possible states. In other words, the state of the right cube not only depends on the state of the left cube, but also on the sequence of moves that put the cube in such a state (and vice versa).

Example. Consider the following permutations.

$$S_1 = (U_1^{\sim})^{-1}\, R_1^{\sim} U_1^{\sim} L_1^{\sim}$$
$$S_2 = L_1^{\sim}\, (U_1^{\sim})^{-1}\, R_1^{\sim} U_1^{\sim}$$

Separating the effects of the permutations on the left and right cube, we have

$$S_{1,\ell} = U_1^{-1} R_1 U_1 L_1 \qquad S_{1,r} = \Delta(U_0^{-1} R_0 U_0 L_0)$$

$$S_{2,\ell} = L_1 U_1^{-1} R_1 U_1 \qquad S_{2,r} = \Delta(L_0 U_0^{-1} R_0 U_0)$$

Calculation shows that $S_{1,\ell} = S_{2,\ell}$, however, $S_{1,r} \neq S_{2,r}$. Therefore, the sequences of turns put the left cube into the same state, but the right cube into different states.

Rotating one or both cubes in space still is not sufficient to hide the moves in $\mathcal{F}$. However, if we allow swapping the cubes, it is. We again define h_1 and h_2 to represent rotations of a single Rubik's cube in space as we did in Section 4.5, except we now do so for the larger 5x5x5 Rubik's cube.

$$h_1 \coloneqq R_0 R_1 \big((3\ 118\ 123\ 51)(8\ 113\ 128\ 56)(17\ 104\ 137\ 65)(22\ 99\ 142\ 70)\big)\, (L_0 L_1)^{-1}$$
$$h_2 \coloneqq U_0 U_1 \big((107\ 83\ 59\ 35)(108\ 84\ 60\ 36)(109\ 85\ 61\ 37)(110\ 86\ 62\ 38)\big)\, (D_0 D_1)^{-1}$$

The effect is the same as depicted in Figure 4.2. Swapping the two cubes is the permutation

$$e_{\leftrightarrow} \coloneqq (1\ 145)(2\ 146)(3\ 147)\ldots(143\ 287)(144\ 288).$$

We construct the repositioning group by allowing simultaneously rotating both cubes in space and swapping the cubes:

$$H \coloneqq \langle \{\, h_1\Delta(h_1), h_2\Delta(h_2), e_{\leftrightarrow} \,\} \rangle.$$

We have $|H| = 48$ since there are $|\langle\{ h_1\Delta(h_1), h_2\Delta(h_2) \}\rangle| = |\langle\{ h_1, h_2 \}\rangle| = 24$ possible rotations of the cubes in space and for each such rotation, we can either simultaneously swap the cubes or not swap the cubes, i.e., $|\langle e_{\leftrightarrow}\rangle| = 2$.
Calculation shows that $f_i^H = \{f_i^h \mid h \in H\} = \mathcal{F}$ for all $f_i \in \mathcal{F}$, therefore, H is indeed a repositioning group of $\mathcal{F}$ (according to Definition 4.1).
To achieve a security of 2^{80}, we can set $d = 45$ since we have $\alpha = 12$.[1] In this case, $r = 926$ rounds are necessary to achieve an impersonation probability below 2^{-30} when $d = 45$. Volte et al. further recommend restricting secret keys such that consecutive permutations do not commute unless they are equal in order to reduce the number of equivalent secret keys [70, p. 85-86], thus reducing the key space to $12 \times 9^{d-1}$ keys since only the first generator can be chosen freely and the remaining generators must either be equal to or not commute with the previous generator. They suggest $d = 48$ and $r = 988$ to account for the smaller key space.

4.7 The puzzle *S*41

The Rubik's cube groups $\mathfrak{R}_3$ (see Section 4.5) and $\mathfrak{R}_5$ (see Section 4.6) are interesting groups for the proposed zero-knowledge protocol. They are also useful for visualizing both the private key as a sequence of turns and especially the respective repositioning groups. The group $\mathfrak{R}_5$, the 5x5x5 Rubik's cube group, even seems to provide enough security for practical use. However, due to the small number of generators $\alpha = |\mathcal{F}|$, the length d of the private key must be quite large to achieve sufficient security, and thus the protocol requires a large numbers of rounds to achieve a sufficiently small impersonation probability.

[1]Volte et al. estimate the security as $d \cdot \alpha^{d/2}$, which is why they suggest $d = 42$ only [70, p. 85], whereas we estimate it to be as low as $2 \cdot \alpha^{d/2}$, leading to slightly higher values for d. In practice, the factor is likely larger.

Therefore, Volte et al. proposed a new puzzle, which they referred to as *S*41 [70, p. 91]. They constructed the puzzle by choosing a single permutation $h \in S_{41}$ and a single permutation $f_1 \in S_{41}$ such that $\left\langle f_1^{\langle h \rangle} \right\rangle = S_{41}$ and attempted to maximize $\alpha = \left| f_1^{\langle h \rangle} \right|$. Their simulation tested 1,000 permutations and resulted in the following parameters.

$$\begin{aligned} h &:= (1\ 14\ 39\ 19\ 31\ 18\ 37)\,(3\ 36\ 4\ 23\ 20\ 34\ 16\ 25\ 17\ 26\ 35)\,(5\ 13\ 30\ 33) \\ &\quad (6\ 7\ 10)\,(8\ 24\ 15\ 38\ 41\ 27\ 11\ 9)\,(12\ 40\ 32\ 21\ 28)\,(22\ 29) \\ f_1 &:= (1\ 11\ 31\ 6\ 17\ 34\ 25\ 24\ 22\ 12\ 4\ 28\ 3\ 14\ 5\ 27\ 32\ 13\ 26\ 8\ 23\ 2\ 20\ 41\ 19\ 10\ 40\ 15\ 38 \\ &\quad 16\ 37\ 39\ 35\ 21\ 18) \\ &\quad (7\ 29\ 36)\,(9\ 30) \end{aligned}$$

Calculation shows that $\alpha = 9240$ [70, p. 91]. Therefore, $d = 12$ is sufficient for a level of security greater than 2^{80}. Due to the small value of d, only $r = \left\lceil 30/\log_2\left(\frac{13}{12}\right)\right\rceil = 260$ rounds are necessary to achieve an impersonation probability of at most 2^{-30}.

4.8 Improving upon *S*41

In the previous section, the puzzle *S*41 was presented with values for h and f_1 suggested by Volte et al. [70] based on a simulation of 1,000 parameter sets. To improve upon *S*41 both within the same symmetric group and in other permutation groups, we attempted to find more efficient or, alternatively, more secure parameters.

4.8.1 Finding permutations of maximal order

What is the difficult part in finding efficient and/or secure parameters? To increase α without increasing the order of the group, a permutation h of higher order must be found within the group. Volte et al. use a computer simulation that randomly selects permutations from S_{41} and selects the permutation which has the highest order. However, we can, at least in theory, find permutations of maximal order based on Landau's function.

Definition 4.9 (Landau's function [38])**.** For $n \in \mathbb{N}$, we define $g(n)$ to be the maximal order of a permutation in the symmetric group S_n,

$$g(n) \coloneqq \max_{\sigma \in S_n} |\langle \sigma \rangle| \,.$$

In general, determining the value of Landau's function for large n is difficult. However, for small values of n, consider the cycle notation of permutations. It is well-known that the order of a permutation is the least common multiple of the length of its disjoint cycles. The set of possible cycle length combinations is the set of integer partitions of n. Therefore, $g(n)$ must be the maximal least common multiple of all integer partitions of n. While there are fewer integer partitions of n than there are permutations in S_n, the complexity of calculating $g(n)$ still grows exponentially.
Once we know $g(n)$ for the desired symmetric group S_n, we can determine the cycle structure of a permutation of order $g(n)$ through prime factorization of $g(n)$. More generally, if we want to find a permutation of order $\alpha \in \mathbb{N}$, we need to consider the prime factorization of α, i.e.,

$$p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k} = \alpha \text{ and } p_1 < p_2 < \cdots < p_k,$$

where $k \in \mathbb{N}$, $p_1, p_2, \ldots, p_k \in \mathbb{P}$, and $e_1, e_2, \ldots, e_k \in \mathbb{N}$. We have

$$\operatorname{lcm}(p_1^{e_1}, p_2^{e_2}, \ldots, p_k^{e_k}) = \alpha$$

because powers of different primes are relatively prime. Thus, a permutation of order α must contain k disjoint cycles, the lengths of which are $c_i \coloneqq p_i^{e_i}$ for $1 \leq i \leq k$. Let $\ell \coloneqq \sum_{1 \leq i \leq k} c_i$ be the number of permuted elements. If $\ell > n$, no such permutation exists in S_n. Otherwise, any permutation

$$(i_{1,1}\ i_{1,2}\ \ldots\ i_{1,c_1})(i_{2,1}\ i_{2,2}\ \ldots\ i_{2,c_2})\ldots(i_{k,1}\ i_{k,2}\ \ldots\ i_{k,c_k})$$

n	Group	Group order	$g(n)$	Prime factorization of $g(n)$
41	S_{41}	$2^{164.5}$	30030	$2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
42	S_{42}	$2^{169.9}$	32760	$2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$
43	S_{43}	$2^{175.3}$	60060	$2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
$\vdots$	$\vdots$	$\vdots$	60060	$2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
47	S_{47}	$2^{197.4}$	120120	$2^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
48	S_{48}	$2^{202.9}$	120120	$2^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
49	S_{49}	$2^{208.6}$	180180	$2^2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
$\vdots$	$\vdots$	$\vdots$	180180	$2^2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
53	S_{53}	$2^{231.3}$	360360	$2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
60	S_{60}	$2^{272.1}$	1021020	$2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$

Table 4.1: Selected symmetric groups and Landau's function value factorization

has order α if the cycles are disjoint, i.e., if

$$\left|\{\, i_{1,1}, i_{1,2}, \ldots, i_{1,c_1}, i_{2,1}, i_{2,2}, \ldots, i_{2,c_2}, \ldots, i_{k,1}, i_{k,2}, \ldots, i_{k,c_k} \,\}\right| = \ell.$$

Note that ℓ might be smaller than n regardless of whether $\alpha = g(n)$ holds.

Example. Recall the puzzle S41 from Section 4.7. We have $n = 41$ and $\alpha = |\langle h \rangle| = 9240$. The prime factorization of α is $2^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 = 9240$. Indeed, h contains cycles of length 8, 3, 5, 7, and 11. However, $g(n) = 30030$, therefore, permutations of higher order exist in S_{41}.

Table 4.1 shows selected symmetric groups beginning with S_{41} and the respective values of Landau's function. For any symmetric group S_n, for which we know $g(n)$, we can easily find a permutation $h \in S_n$ with $|\langle h \rangle| = g(n)$ now. Then, we only need to select $f \in S_n$ such that $\langle \mathcal{F} \rangle = S_n$, where $\mathcal{F} = f^{\langle h \rangle}$.

4.8.2 Nothing-up-my-sleeve permutations

In the past, where algorithms required fixed seed values, these were often chosen by the proposing entity. Even if said entity might claim that the parameter was chosen randomly, there is no guarantee that the proposed parameters do not enable some backdoor that is only known to the party that selected the values. Therefore, it has become common practice to instead derive seed values from constants that are unlikely to have any hidden properties, and in a way that is as simple as possible. Thereby derived parameters are often referred to as *nothing-up-my-sleeve numbers*.

Here, we will develop a deterministic algorithm that produces h and f_1 with desirable properties as discussed in the previous section for any group S_n. We derive h from the fractional part of the constant π, the ratio of the circumference of a circle to its diameter, and f_1 from the fractional part of the constant e (Euler's number). The construction is straightforward and is not affected by any parameters other than n.[2]

Algorithm 4.5 defines the produce sequence function, which converts a real number's fractional part to a sequence of n integers, such that the first integer is in $\{ 1, 2, \ldots, n \}$ for some $n \in \mathbb{N}$, the second integer is in $\{ 1, 2, \ldots, n-1 \}$, the third integer is in $\{ 1, 2, \ldots, n-2 \}$, etc., and the n-th integer is always 1. Therefore, it produces one of $n!$ possible sequences. The function is defined recursively. Unless $n = 0$, the function does not discard any information, meaning that it maintains the full entropy of the real number v until the entire integer sequence has been generated.

Algorithm 4.5 Parameter generation: the produce sequence function

```
Input: $n \in \mathbb{N}_0, v \in [0, 1)$
Output: $s \in (\{ 1, 2, \ldots, n \} \times \{ 1, 2, \ldots, n-1 \} \times \cdots \times \{ 1, 2 \} \times \{ 1 \})$
1: function PRODUCE SEQUENCE($n$, $v$)
2:     if $n = 0$ then return empty sequence
3:     $p \leftarrow 1 + n \cdot v$
4:     $r \leftarrow \lfloor p \rfloor$
5:     return CONCAT($r$, PRODUCE SEQUENCE($n - 1, p - r$))
```

[2]However, for large n, it might be desirable to fix α to a "small" value.

The produce shuffle function uses the integer sequence that is generated by the above produce sequence function to shuffle the integers from one to n. Note that this is a one-to-one mapping since the sequence is one of $n!$ and the shuffled output is one of $n!$.

Algorithm 4.6 Parameter generation: the produce shuffle function

```
Input: $n \in \mathbb{N}$, $v \in [0, 1)$
Output: $s \in \{1, 2, \dots, n\}^n$, such that each $1 \leq i \leq n$ occurs in $s$ exactly once
1: function produce shuffle($n$, $v$)
2:     $remaining \leftarrow 1, 2, \dots, n$
3:     $s \leftarrow$ empty sequence
4:     for $i$ in produce sequence$(n, v)$ do
5:         $s \leftarrow$ concat$(s, remaining[i])$
6:         delete $remaining[i]$
7:     return $s$
```

The produce permutation function uses produce shuffle to obtain the shuffled list of integers from one to n and forms disjoint cycles according to the prime factorization of α.

Algorithm 4.7 Parameter generation: the produce permutation function

```
Input: $n \in \mathbb{N}$, $\alpha \in \mathbb{N}$, $v \in [0, 1)$
Output: $\sigma \in S_n$
1: function produce permutation($n$, $\alpha$, $v$)
2:     $\sigma \leftarrow \mathrm{id}_{S_n}$
3:     shuffled $\leftarrow$ produce shuffle$(n, v)$
4:     $p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k} \leftarrow$ prime factorization of $\alpha$
5:     for $j \leftarrow 1$ to $k$ do
6:         $c_j \leftarrow p_j^{e_j}$
7:         $i_1, i_2, \dots, i_{c_j} \leftarrow$ take next $c_j$ numbers from shuffled
8:         $\sigma \leftarrow \sigma \circ (i_1\ i_2\ \dots\ i_{c_j})$
9:     return $\sigma$
```

Finally, the produce parameters function uses produce permutation to generate $h \in S_n$ from the fractional part of π. It then does the same to generate $f_1 \in S_n$ from the fractional part of e. While this works for most n, there is no guarantee that $\left\langle f_1^{\langle h \rangle} \right\rangle = S_n$. Therefore, if this requirement is not met, the algorithm (repeatedly) discards the first decimal digit of the fractional part of e until the condition is true.

Note that an implementation of these algorithms requires floating-point arithmetic with much higher precision than usually provided by hardware. Arbitrary precision libraries,

Algorithm 4.8 Parameter generation: the produce parameters function

```
Input: $n \in \mathbb{N}$
Output: $(h, f_1) \in S_n \times S_n$, such that $\left\langle f_1^{\langle h \rangle} \right\rangle = S_n$ and $|\langle h \rangle| = g(n)$
1: function PRODUCE PARAMETERS($n$)
2:     $\alpha \leftarrow g(n)$                                      ▷ $g$ is Landau's function
3:     $h \leftarrow$ PRODUCE PERMUTATION($n, \alpha, \pi - \lfloor \pi \rfloor$)     ▷ $\pi$ is Archimedes's constant
4:     $e' \leftarrow e$                                             ▷ $e$ is Euler's number
5:     repeat
6:         $f_1 \leftarrow$ PRODUCE PERMUTATION($n, \alpha, e' - \lfloor e' \rfloor$)
7:         $e' \leftarrow 10 \cdot e'$
8:     until $\left\langle f_1^{\langle h \rangle} \right\rangle = S_n$
9:     return $(h, f_1)$
```

such as `RealField` in SageMath [68], can be used to achieve the required precision of hundreds or thousands of digits (depending on n).

4.8.3 New parameter sets

Based on the discussion in the previous sections, we now propose multiple new parameter sets as alternatives to the puzzle *S*41 (see Section 4.7).

***S*41**$*$ is the puzzle obtained by produce parameters(41).

$$\begin{aligned} h &:= (6\ 34)\,(10\ 13\ 5)\,(9\ 4\ 39\ 27\ 40)\,(24\ 36\ 20\ 21\ 3\ 28\ 19) \\ &\quad (16\ 14\ 7\ 17\ 38\ 12\ 32\ 15\ 23\ 30\ 2)\,(29\ 31\ 37\ 1\ 33\ 26\ 8\ 11\ 25\ 22\ 18\ 35\ 41) \\ f_1 &:= (30\ 18)\,(41\ 12\ 25)\,(36\ 31\ 14\ 1\ 4)\,(5\ 20\ 34\ 11\ 10\ 15\ 3) \\ &\quad (29\ 22\ 21\ 17\ 33\ 38\ 37\ 23\ 16\ 13\ 24)\,(9\ 32\ 40\ 6\ 39\ 19\ 7\ 27\ 2\ 8\ 28\ 26\ 35) \end{aligned}$$

*S*41$*$ improves upon *S*41 within the same symmetric group due to the larger value of $\alpha = 30030$ and the deterministic process through which its parameters were obtained. To achieve a security of 2^{80}, a secret key length of only $d = 11$ is necessary (as opposed to 12 for *S*41). Consequently, for an impersonation probability below 2^{-30}, only 239 rounds are necessary (as opposed to 260 rounds for *S*41).

***S*43∗** is the puzzle obtained by produce parameters(43).

$$\begin{aligned}
h &:= (7\ 4\ 32\ 17)\,(35\ 27\ 23)\,(28\ 37\ 15\ 38\ 29)\,(34\ 6\ 30\ 36\ 3\ 41\ 19)\\
&\quad (43\ 16\ 9\ 18\ 40\ 10\ 13\ 31\ 33\ 8\ 42)\,(1\ 22\ 24\ 39\ 21\ 5\ 12\ 25\ 20\ 2\ 26\ 14\ 11)\\
f_1 &:= (31\ 39\ 9\ 38)\,(36\ 7\ 37)\,(41\ 32\ 20\ 22\ 14)\,(12\ 35\ 17\ 19\ 21\ 40\ 29)\\
&\quad (3\ 26\ 4\ 1\ 13\ 27\ 24\ 6\ 11\ 33\ 5)\,(2\ 23\ 28\ 10\ 43\ 30\ 34\ 8\ 18\ 16\ 25\ 15\ 42)
\end{aligned}$$

We chose *S*43∗ because it allows further reducing d to $d = 10$ for the same level of security, leading to only $r = 219$ rounds for an impersonation probability below 2^{-30} while not increasing the size of each permutation much. We have $\alpha = 60,060$.
***S*53∗** is the puzzle obtained by produce parameters(53).

$$\begin{aligned}
h &:= (8\ 28\ 13\ 38\ 46\ 37\ 17\ 39)\,(27\ 35\ 6\ 42\ 10\ 11\ 1\ 26\ 18)\,(33\ 34\ 36\ 9\ 21)\\
&\quad (3\ 50\ 30\ 49\ 25\ 19\ 12)\,(45\ 53\ 29\ 24\ 32\ 20\ 48\ 43\ 47\ 23\ 7)\\
&\quad (40\ 15\ 52\ 44\ 16\ 31\ 41\ 2\ 4\ 14\ 51\ 22\ 5)\\
f_1 &:= (39\ 4\ 31\ 45\ 2\ 36\ 6\ 22)\,(53\ 25\ 38\ 5\ 8\ 16\ 11\ 26\ 37)\,(35\ 18\ 30\ 51\ 50)\\
&\quad (15\ 48\ 21\ 49\ 20\ 42\ 46)\,(41\ 13\ 34\ 33\ 43\ 47\ 14\ 44\ 29\ 32\ 10)\\
&\quad (9\ 27\ 28\ 23\ 17\ 19\ 24\ 40\ 12\ 7\ 3\ 1\ 52)
\end{aligned}$$

Lastly, we chose *S*53∗ because it appears to be a good candidate for achieving a security of approximately 2^{112}, which is a recommended lower bound until 2030 [4]. We have $\alpha = 360360$. Using $d = 12$ and $r = 260$ as in the *S*41 puzzle suggested by Volte et al. [70], the security is estimated to be *at least* $2^{111.8}$ and the impersonation probability is below 2^{-30}. The increase in the size of permutations, compared to *S*41, does not impact performance much.

Remark 4.10. Note that the meet-in-the-middle attack (see Theorem 3.14), which the security estimation is based on, requires the attacker to store approximately $2^{110.8}$ permutations, or approximately $2^{35.6}$ yobibytes.

To generate parameters with even higher levels of security, the PRODUCE PARAMETERS algorithm can also be used for $n > 53$. For much larger values of n, the parameter α can be set to "small" values instead of setting it to $g(n)$, which grows exponentially. Only line 2 in Algorithm 4.8 needs to be adapted accordingly.

4.9 Implementation considerations

Assuming that a software implementation of the required permutation operations is available, implementing the zero-knowledge protocol (see Section 4.3) is straightforward. A reference implementation written in the programming language Python 3 is provided in Appendix B. It is sufficiently abstract to be compatible with any of the previously described parameter sets, i.e., the 3x3x3 Rubik's cube (see Section 4.5), the 5x5x5 Rubik's cube (see Section 4.6), the puzzle *S*41 (see Section 4.7), and the new parameter sets proposed in this thesis (see Section 4.8).

However, for practical use, it is insufficient for a software implementation to merely be correct. Performance and security of the implementation are similarly important. The security of the implementation is not to be confused with the security of the protocol itself. The protocol, including its specification and parameters, might be secure (in theory) and yet a correct implementation might not be.

We have made an implementation of the protocol that is written in the C programming language and that considers the following difficulties available online at

`https://github.com/tniessen/zkp-volte-patarin-nachef-c`.

4.9.1 Random number generation

In this case, the protocol requires both the prover and the verifier to make a multitude of random choices. Random number generation is a traditionally difficult problem for computers that are designed to work deterministically. Due to the limited availability of randomness, software often relies on *cryptographically secure pseudorandom number generators*. Modern computer architectures often provide access to hardware features, e.g., using the `RDRAND` CPU instruction, which uses a cryptographically secure hardware random number generator [27].

Even when a secure random number generator is available, its use must be designed carefully. For example, when generating a secret key (see Algorithm 4.1, the prover must choose d random integers from the set $\{ 1, \ldots, \alpha \}$. However, random number generators usually produce random integers from a set whose cardinality is a power of two. For example, a typical random number generator RAND might produce values with uniform probability from the set $\{ 0, \ldots, 2^b - 1 \}$ for some integer $b > 0$. While naive implementations, such as $i_j := 1 + (\text{RAND} \bmod \alpha)$, appear to work, they do not produce a uniform probability distribution unless 2^b is a multiple of α, and implementations must account for this so-called *modulo bias* accordingly.

A standard work on random number generation is Knuth's *Seminumerical Algorithms* [31]. A few mechanisms for obtaining cryptographically secure random numbers from sources of randomness with limited entropy are approved by the *National Institute of Standards and Technology* [5].

4.9.2 Random permutation generation

Randomly choosing permutations from a permutation group can also be difficult, especially when a uniform probability distribution is desired. In the proposed zero-knowledge protocol, the prover must pick $\tau \in_R H$ (in line 1 of Algorithm 4.2) and $\sigma_0 \in_R G'$ (in line 2 of Algorithm 4.2).

In all previously discussed parameter sets, the group H is small enough to list all of its elements, allowing to select a random element τ by choosing a random index within the list of elements of H. Refer to the previous section for producing random integers.

Recall that $G' = \langle \mathcal{F}, H \rangle$. Fortunately, some of the suggested parameter sets yield $G' = S_n$ for some $n \in \mathbb{N}$. We have $G' = S_{41}$ for the puzzles $S41$ (see Section 4.7) and $S41*$ (see Section 4.8), $G' = S_{43}$ for the puzzle $S43*$ and $G' = S_{53}$ for $S53*$ (see Section 4.8). Choosing a random permutation from a symmetric group S_n is straightforward: write

the identity id_{S_n} in the two-line notation, then apply the Fisher–Yates shuffle [31, p. 145] to the second line.
However, when G' is not a symmetric group (e.g., in Section 4.5 and Section 4.6), producing random elements of G' is more difficult. Let S_n be the symmetric group for some $n \in \mathbb{N}$ such that $G' < S_n$. It is possible to repeatedly choose random permutations from S_n as described above until one of the generated permutations is in G'. However, merely deciding if the generated permutation is in G' is potentially difficult (see Definition 3.9). Even if the structure of G' allows making this decision efficiently, G' is, with high probability, much smaller than S_n, which makes it very unlikely that a randomly selected element of S_n is in G'.

Example. The parameters for the 3x3x3 Rubik's cube (see Section 4.5) result in $|G'| \approx 8.65 \times 10^{19}$, but $|S_{48}| \approx 1.24 \times 10^{61}$. Therefore, the probability that a random element of S_{48} is also an element of G' is only 6.97×10^{-42}.

Thus, a constructive approach appears to be the best option. Because $G' = \langle \mathcal{F}, H \rangle$, randomly selecting a finite number of not necessarily distinct elements $g_1, g_2, \ldots, g_m \in_R \mathcal{F} \cup H$ and computing their composition $g = g_1 g_2 \ldots g_m$ yields a random element of G'. However, the probability distribution is not uniform, meaning that, for any value of m, some elements of G' will be more likely than others. Nevertheless, for sufficiently large values of m, the probability distribution approximates a uniform distribution.
To select m not necessarily distinct random elements of $\mathcal{F} \cup H$, use the method for selecting $\tau \in_R H$ that was described above and adapt it accordingly.

4.9.3 Side-channel vulnerabilities

Side-channel attacks are attacks that do not target weaknesses of the cryptographic algorithm itself, but instead use weaknesses of the implementation. The most prominent of such attacks are *timing attacks* and *cache* (*timing*) *attacks*, many of which are based on work by Kocher from as early as 1996 [34].

If the duration of a computation within a software or hardware implementation depends on potentially sensitive data, an attacker might be able to draw conclusions about the data from measuring the duration of the computation directly or indirectly. Such attacks are referred to as *timing attacks*.

Writing software that is resistant to timing attacks proves to be challenging. Algorithms and implementations are usually designed and optimized to be as fast as possible and not to run in constant time. Even if the software itself appears to not contain any timing vulnerabilities, the compiler (or interpreter) could introduce side-channel vulnerabilities, e.g., due to optimization of the generated instructions. And even if the code is timing-safe, the system hardware might execute the same instruction sequences differently depending on various other conditions.

Cache (timing) attacks analyze memory access patterns indirectly. Modern computer processors represent the system's memory as a sequence of pages of a fixed size and store the contents of frequently accessed memory pages in caches. This technique significantly improves performance because accessing data that is stored within a cache is much faster than accessing data that is stored in the system's regular main memory. However, caches are small memory buffers, which are very limited in size (usually in the order of kibibytes or mebibytes), therefore, the system evicts memory pages from its cache if it determines that caching other memory pages is more likely to improve the system's performance.

Depending on whether a memory page is currently cached, an attacker might observe different memory access times. Therefore, by measuring the delay incurred by memory access to specific memory pages, an attacker can sometimes determine which memory pages have previously been accessed by other processes and thus cached by the CPU. A similar technique played an important role in the recently discovered *Meltdown* [39] and *Spectre* [33] vulnerabilities that affect a wide range of applications and systems.

An implementation of the zero-knowledge protocol could be affected by cache timing vulnerabilities. While most of the required data structures are small, some parameter sets, e.g., $S41$ (see Section 4.7) and the new puzzles $S41*$, $S43*$ and $S53*$ (see Section 4.8), use a large number of permutations as the respective repositioning group H and as the set of generators $\mathcal{F}$. Generating H "on the fly," that is, only when required, appears to be challenging in constant time, i.e., without introducing timing side-channel vulnerabilities. Similarly, while $f_i \in \mathcal{F}$ can be computed in constant time for the aforementioned puzzles, it might still make sense to store $\mathcal{F}$ in memory to improve performance. If all elements of H and $\mathcal{F}$ have been computed in advance and are stored in memory, at least $|H| \cdot \lceil \log_2(|H|) \rceil + |\mathcal{F}| \cdot \lceil \log_2(|G_R|) \rceil$ bits are required, i.e., at least $1,653,960$ bits for $S41$, $5,405,400$ bits for $S41*$, $11,531,520$ bits for $S43*$, and $90,450,360$ bits for $S53*$. However, this minimal representation makes operations that use permutations difficult to implement in software, which is why software implementations usually represent elements of subgroups of a symmetric group S_n as sequences of n integers. This representation in memory is equivalent to the mathematical two-line notation of permutations, where the first line corresponds to the (relative) location in memory.

$$\begin{pmatrix} 1 & 2 & \dots & n-1 & n \\ \pi(1) & \pi(2) & \dots & \pi(n-1) & \pi(n) \end{pmatrix}$$

For example, a software implementation would usually represent $\pi \in S_{41}$ as a sequence of the 41 integers $\pi(1), \pi(2), \dots, \pi(41)$. This representation increases the memory requirement for storing all elements of $H \leq S_n$ to $|H| \cdot n$ integers, and similarly for $\mathcal{F}$. Regardless of the representation of the permutations, the required amount of memory is significantly larger than a typical memory page size of 4096 bytes. During key generation, for example, the prover might only access a few of the stored permutations. Because the permutations are stored across a multitude of memory pages, only a small number of memory pages and, thus, only a subset of the permutations might be placed in a CPU cache. In that case, a cache timing attack might succeed in determining which of the permutations have been used by the prover.

For instance, with the $S41$ parameter set (see Section 4.7), $\alpha = 9240 = |H| = |\mathcal{F}|$. While generating the public key, the prover must access f_{i_j} for $1 \leq j \leq d$, where $(i_1, \dots, i_d) \in$

$\{ 1, \dots, \alpha \}$ is the private key, or, if only H is stored in memory, must access h_{i_j} to compute f_{i_j}. Assuming a typical page size of 4096 bytes and that each permutation is stored as a sequence of 41 bytes, the algorithm might access only $d = 12$ of $\lceil 41 \cdot 9240/4096 \rceil = 93$ memory pages holding F (or H). Those 12 pages only hold approximately 1200 of the 9240 permutations, thus knowing which pages have been accessed, e.g., through a cache timing attack, reduces the key space from $\alpha^d \approx 2^{158}$ to $1200^d \approx 2^{122}$.
Therefore, instead of storing m elements of a group S_n as a sequence of m sequences of n integers each, we suggest interleaving the m sequences. In other words, in place of a memory layout such as

$$\begin{bmatrix} f_1(1) & f_1(2) & \dots & f_1(n) & f_2(1) & f_2(2) & \dots & f_2(n) & \dots & f_m(1) & f_m(2) & \dots & f_m(n) \end{bmatrix},$$

we interleave all permutations to maximize the number of permutations that cause each memory page to be accessed:

$$\begin{bmatrix} f_1(1) & f_2(1) & \dots & f_m(1) & f_1(2) & f_2(2) & \dots & f_m(2) & \dots & f_1(n) & f_2(n) & \dots & f_m(n) \end{bmatrix}$$

Since $\alpha > 4096$, no memory page contains more than one byte of each permutation. Consequently, each permutation is spread out across $n = 41$ different memory pages, and accessing a single permutation gives each of the 93 memory pages a chance of $\frac{41}{93}$ of being accessed. Accessing $d = 12$ permutations during key generation thus reduces the probability that a specific memory page is not accessed to $\left(1 - \frac{41}{93}\right)^{12} \approx 0.09\%$, which means that every memory page has a 99.91% probability of being accessed.
Of course, this modified layout makes memory access slower and the CPU cache significantly less effective. However, it also appears to render cache timing attacks targeting memory pages that hold large numbers of permutations, e.g., H or $\mathcal{F}$, ineffective.

4.9.4 Transmitting τ and $f_{i_q}^{\tau}$

A minor optimization can be implemented in order to reduce the amount of data transferred between prover and verifier.

Instead of sending $\tau \in H$ to the receiver, assign fixed indices to all elements of H, such that $H = \{ h_1, h_2, \ldots, h_{|H|} \}$, and only transfer the index $i \in \{ 1, 2, \ldots, |H| \}$ such that $h_i = \tau$. Instead of checking $\tau \in H$, the verifier only needs to check $1 \leq i \leq |H|$.
Similarly, $f_{i_q}^{\tau}$ is an element of $\mathcal{F}$ and there already exists a numbering of the elements of $\mathcal{F} = \{ f_1, f_2, \ldots, f_\alpha \}$. Therefore, instead of transmitting $f_{i_q}^{\tau}$, the prover only needs to send $j \in \{ 1, 2, \ldots, \alpha \}$ such that $f_j = f_{i_q}^{\tau}$, and the verifier only needs to check $1 \leq j \leq \alpha$.

4.10 Outlook

In this chapter, we have discussed an interesting zero-knowledge scheme due to Volte et al. [70]. Two constructions based on such an interactive zero-knowledge protocol are of particular interest.
The protocol requires the prover and verifier to interact during each of potentially hundreds of rounds, which, in reality, could incur serious communication delays, far outweighing any computational cost. To overcome this problem, a standard construction such as the one developed by Blum et al. [6] can likely be used to create a *non-interactive* zero-knowledge system.
Secondly, Fiat and Shamir developed a method for constructing digital signature schemes from interactive proof systems [19]. This mechanism has similarities to the construction due to Blum et al. [6]. Thus, the zero-knowledge protocol discussed here can likely be used to construct a digital signature scheme.

Chapter 5

A symmetric encryption scheme

Few false ideas have more firmly gripped the minds of so many intelligent men than the one that, if they just tried, they could invent a cipher that no one could break.
— David Kahn, *The Codebreakers*

In 2020, Pan et al. proposed two symmetric encryption schemes based on Rubik's cubes. In particular, the schemes are based on the assumption that, if one draws an arrow pointing in one of four possible directions onto each facelet of a Rubik's cube[1] and then performs a sufficiently long sequence of turns on the Rubik's cube, it is infeasible to draw conclusions about the original arrow orientations from the resulting arrow orientations alone [48]. Of course, if one knew the sequence of turns that had been applied to the cube, one could reverse the sequence and obtain the original arrow orientations. Similarly, if one knew the state of the Rubik's cube after the sequence had been applied, one could simply solve the Rubik's cube (in any number of moves) and thus obtain the original arrow orientations. However, knowing only the final direction of the arrows on the facelets of the cube, without knowing the original positions of said facelets, it does seem almost impossible to obtain their original orientations.

[1]Interestingly, such puzzles have existed since shortly after the Rubik's cube's invention [56, p. 195].

This idea seems intriguing especially because Pan et al. claim that the design is provably secure based on the intractability assumption of the conjugacy problem for the Rubik's cube [48]. Goldwasser et al. wrote in 1989: "Given our current state of knowledge about lower bounds, the security of a cryptographic protocol must be proved based on the intractability assumption of some candidate hard problem. Thus one must accept that further analysis may reveal some candidate hard problems to be efficiently solvable. What is not acceptable is that a protocol may be broken without violating the relative intractability assumption" [23, p. 301]. However, this chapter will thoroughly disprove the security properties claimed by Pan et al. [48] and thus break the proposed schemes without violating the intractability assumption of the conjuacy problem. We will not only disprove the security property but even present practical attacks. To the best of the author's knowledge, this is the first cryptanalysis of the proposed schemes.

5.1 Message encoding

Unlike many other models that do not use the center facelets, the proposed schemes use all 54 facelets of a standard Rubik's cube. While the positions of the center facelets do not change when turning the faces of the cube, their orientation does change.

A two-dimensional representation of the surface of the Rubik's cube trivially allows arrows pointing in four directions: up, right, down, and left. These four possible directions can be encoded using two bits each. We choose to represent the arrows on the facelets of the Rubik's cube as elements of $\mathbb{Z}/4\mathbb{Z}$, thus the 54 arrows on the 54 facelets of the cube are in $(\mathbb{Z}/4\mathbb{Z})^{54}$. Instead of defining encoding and decoding functions, we will use $\uparrow$ synonymously with $0 \in \mathbb{Z}/4\mathbb{Z}$, $\rightarrow$ synonymously with $1 \in \mathbb{Z}/4\mathbb{Z}$, $\downarrow$ synonymously with $2 \in \mathbb{Z}/4\mathbb{Z}$, and $\leftarrow$ synonymously with $3 \in \mathbb{Z}/4\mathbb{Z}$.

No encoding or decoding is necessary in either direction. The binary representation of $(\mathbb{Z}/4\mathbb{Z})^{54}$ is $\{0,1\}^{108}$. Position changes of the arrows are simply permutations of the 54 dimensions, and orientation changes are additions in $\mathbb{Z}/4\mathbb{Z}$:

$$((((\uparrow)+1)+1)+1)+1 = (((\rightarrow)+1)+1)+1 = ((\downarrow)+1)+1 = (\leftarrow)+1 = (\uparrow).$$

Therefore, the bit sequence

$$\mathsf{00\,01\,10\,11}\ldots\mathsf{00\,01\,10\,11} \in \{0,1\}^{108}$$

represents

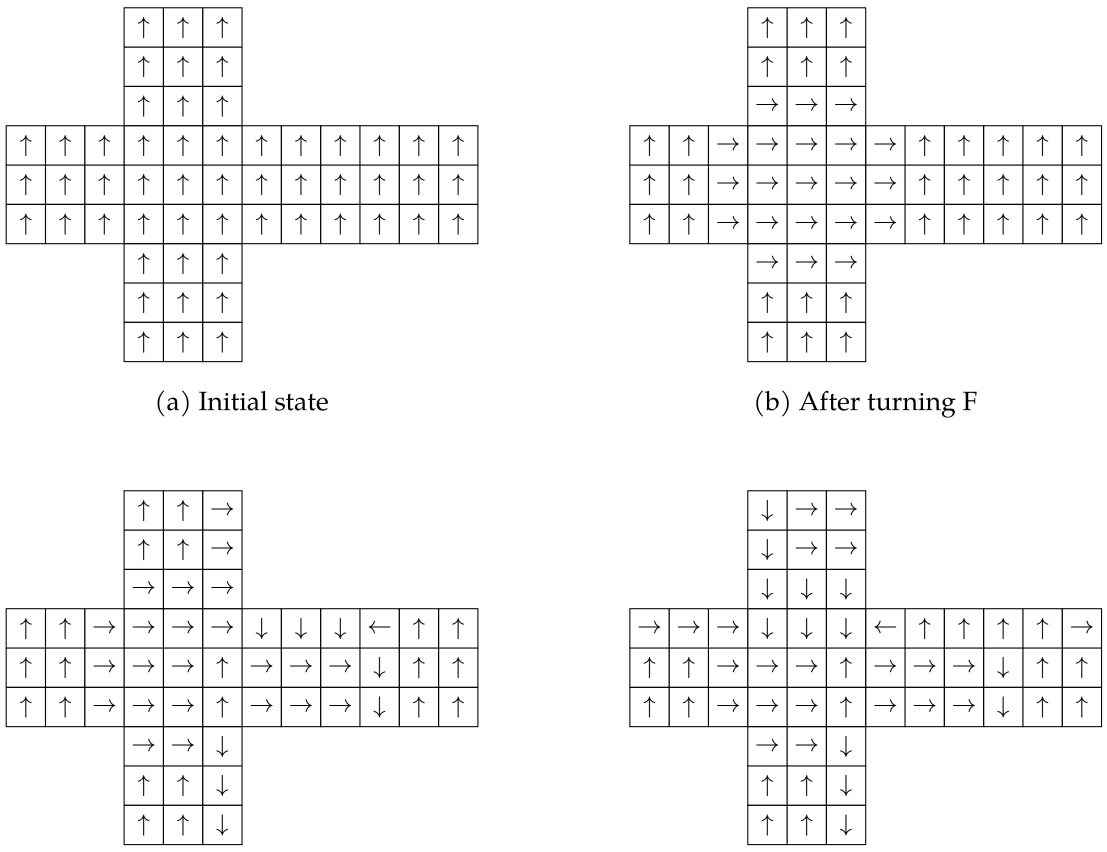
$$(0,1,2,3,\ldots,0,1,2,3) \in (\mathbb{Z}/4\mathbb{Z})^{54},$$

which is *equal* to

$$(\uparrow,\rightarrow,\downarrow,\leftarrow,\ldots,\uparrow,\rightarrow,\downarrow,\leftarrow).$$

Because the scheme proposed by Pan et. al., by design, makes no use of the colors of the facelets [48], $(\mathbb{Z}/4\mathbb{Z})^{54}$ is also a representation of the state of the Rubik's cube at any point in time. In other words, the only information associated with each facelet is the direction of the arrow on it, and not its color or its original position.

Figure 5.1 shows the effects of a simple turn sequence consisting of three clockwise quarter turns only, beginning with the initial state $(0,0,\ldots,0,0) \in (\mathbb{Z}/4\mathbb{Z})^{54}$. Given the orientation of the arrows in Figure 5.1d alone, it is certainly possible to restore the original orientation of the arrows by applying the sequence $U^{-1}R^{-1}F^{-1}$. However, knowing only that a sequence of length d was applied, the correct sequence to restore the arrows to their original orientations is just one of 6^d possible sequences of clockwise quarter turns. In this example, that only leaves 216 possibilities, however, the number grows exponentially with the length of the sequence.



(a) Initial state

(b) After turning F

(c) After turning F and R

(d) After turning F, R, and U

Figure 5.1: Example sequence of facelet orientations

5.2 Algorithms

The scheme $\mathfrak{S}_1$ defines a block cipher operating on 108 bit blocks, but due to the way messages are encoded, we prefer to work in the message space $\mathcal{M} = (\mathbb{Z}/4\mathbb{Z})^{54}$. During the encryption process, a random rotating sequence $r \in \mathfrak{R}_3$ is used, which is appended to the ciphertext. Therefore, the ciphertext space is $\mathcal{C} = \mathcal{M} \times \mathfrak{R}_3$. The symmetric key is also a random rotating sequence, therefore, the key space is $\mathfrak{R}_3$.

- **Key generation** is as simple as choosing the secret key $k \in_R \mathfrak{R}_3$ randomly (as a sequence of clockwise quarter turns by default).
- To **encrypt** a message $m = (m_1, \ldots, m_{54}) \in \mathcal{M}$ with a secret key $k \in \mathfrak{R}_3$, select $r \in \mathfrak{R}_3$ randomly and assign m_1, …, m_{54} to the facelets at position 1, 2, …, 54 of a Rubik's cube, respectively, then perform the turn sequence $k^{-1}rk$ on the Rubik's cube. Let m'_1, m'_2, …, m'_{54} be the resulting arrow orientations on the facelets at position 1, 2, …, 54, respectively, and let $m' = (m'_1, m'_2, \ldots, m'_{54})$.
  The ciphertext is (m', r).
- To **decrypt** a ciphertext $(m', r) \in \mathcal{M} \times \mathfrak{R}_3$ with a secret key $k \in \mathfrak{R}_3$, where $m' = (m'_1, m'_2, \ldots, m'_{54})$, assign m'_1, m'_2, …, m'_{54} to the facelets at position 1, 2, …, 54 of a Rubik's cube, respectively, then perform the turn sequence $k^{-1}r^{-1}k$ on the Rubik's cube. Let m_1, m_2, …, m_{54} be the resulting arrow orientations on the facelets at position 1, 2, …, 54, respectively, and let $m = (m_1, m_2, \ldots, m_{54})$. The plaintext is m.

5.3 Breaking the scheme through the cube's centers

A flaw in the proposed scheme arises from the inclusion of the center facelets in the scheme. None of the permutations U, L, F, R, D, and B change the positions of the center facelets, and the orientation of each center facelet (or, equivalently, of the arrow thereon) only depends on turns of the same face. For example, the orientation of the center facelet of the upper face of the cube only changes when the U turn is applied to the cube.

Theorem 5.1. *The symmetric encryption scheme $\mathfrak{S}_1$ allows recovering a significant part of the plaintext without knowing the secret key.*

Proof. Let k be the secret key, and let y be the number of clockwise quarter turns of U in k. During encryption, a random turn sequence r is chosen. Let z be the number of clockwise quarter turns of U in r.
Ignoring all facelets other than the center facelet of U, applying k^{-1} followed by r followed by k is equivalent to applying U^y followed by U^z followed by U^{-y}. The order of these operations is irrelevant, and since counterclockwise turns nullify subsequent clockwise turns of the same face, this sequence is equivalent to U^z as long as we only consider the center facelet of U. Unlike the secret key k, the random sequence r and, therefore, z are public. All that is necessary to recover the original orientation of the center facelet of the upper face is applying U^{-z} to the Rubik's cube.
Therefore, by repeating this procedure for each of the six faces, it is trivially possible to restore the original orientations of the center facelets and, since the position of a facelet on the surface of the Rubik's cube maps directly to the position of the bits assigned to it in the plaintext and ciphertext, to recover the parts of the plaintext that are assigned to center facelets. Since there are six center facelets and each represents two bits of data, 12 of the 108 bits can be recovered without knowing the symmetric key. □

The implications of including the center facelets in the permutation group were mentioned by David Singmaster even before 1981, when he referred to the extension of $\mathfrak{R}_3$ to the center facelets and their orientations as *the supergroup* [66, p. 18].

Example. Consider the example given by Pan et al. [48]. The plaintext is

$$\begin{aligned} m = & \texttt{111001100011000011010111110000001001} \\ & \texttt{001001111001011000011010111000000111} \\ & \texttt{011010001111101100011110010010110100}. \end{aligned}$$

We intentionally do not present the encryption key here. The random sequence r is $r = RLFBUDRFBU$, and the ciphertext is

$$
\begin{aligned}
c = & \texttt{111111001011110000011110111001001111} \\
& \texttt{101101100011001011011000010111000011} \\
& \texttt{100101111001101110011010101001000100}.
\end{aligned}
$$

The center facelet of the upper face is the fifth facelet of the cube, therefore, it corresponds to the ninth and tenth bit of the decoded message. Counting the number of clockwise quarter turns of U in r, we see that U was rotated twice during the application of r. The ninth and tenth bits have the values 1 and 0, respectively, which means that the arrow on the facelet is pointing down ($\downarrow$) as described in Section 5.1. After applying two counterclockwise turns of U to a Rubik's cube in any state that shows $\downarrow$ on the center facelet of the upper face, the arrow on said facelet now points into the opposite direction ($\uparrow$), which represents the two bits 00. Comparing this result with the plaintext, we see that the ninth and tenth bits of the plaintext are indeed 0 and 0, respectively.

It is possible to modify the scheme by removing the center facelets from the model, which makes the previously described attack impossible, but also reduces the block size from 108 bits to 96 bits. However, as the next section will show, even this modified scheme is provably insecure.

5.4 Disproving IND-CPA and IND-CCA2 security

We will now develop an attack that, regardless of the previously discovered flaw due to the center facelets, disproves IND-CPA security of the scheme $\mathfrak{S}_1$ and consequently the IND-CCA2 security of the scheme $\mathfrak{S}_2$, which are due to Pan et al. [48].

Definition 5.2. Let $s_\delta \colon \mathcal{M} \to \{\, 0, 1 \,\}$ be defined as

$$s_\delta(m) := \left(\sum_{i \in \mathcal{I}} m_i \right) \bmod 2,$$

where $m = (m_1, m_2, \ldots, m_{54}) \in \mathcal{M}$ and where $\mathcal{I} = \{\, 1, \ldots, 54 \,\} \setminus \{\, 5, 14, 23, 32, 41, 50 \,\}$ is the set of all non-center facelet indices.

Remark 5.3. $m_i \bmod 2 = 1$ if and only if m_i is perpendicular to $\uparrow$.

Let $\operatorname{turn} \colon \mathcal{M} \times \{\, U, L, F, R, B, D \,\} \to \mathcal{M}$ be the function that maps a state and a face to the state that results from turning said face of a Rubik's cube in said previous state.
Let $\operatorname{turn}^* \colon \mathcal{M} \times \{\, U, L, F, R, B, D \,\}^* \to \mathcal{M}$ be the function that applies the turn function repeatedly in order to apply a sequence of turns.

Lemma 5.4. *Let $m \in \mathcal{M}$ be a state and $Q \in \{\, U, L, F, R, B, D \,\}$ be a face.*
Then $s_\delta(m) = s_\delta(\operatorname{turn}(m, Q))$.

Proof. First, observe that each possible turn moves facelets from positions in the set $\mathcal{I}$ only to other positions within the same set $\mathcal{I}$. Center facelets remain in the centers of their respective faces, and no turn will move a facelet whose position is in $\mathcal{I}$ to a central position.
Let $m = (m_1, m_2, \ldots, m_{54})$ and let $m' = (m'_1, m'_2, \ldots, m'_{54})$, where $m' := \operatorname{turn}(m, Q)$.
Let $\mathbf{R}_{i,Q} \in \mathbb{Z}/4\mathbb{Z}$ be the value in row $1 \leq i \leq 54$ and column Q of Table 5.1. $\mathbf{R}_{i,Q}$ is the relative orientation change of the arrow on the i-th facelet that is incurred by a clockwise quarter turn of a face Q. Because facelet positions also change, $m_i = m'_i + \mathbf{R}_{i,Q}$ is generally *not* true.

i	U	L	F	R	B	D
1	1	0	0	0	3	0
2	1	0	0	0	3	0
3	1	0	0	2	3	0
4	1	0	0	0	0	0
5	1	0	0	0	0	0
6	1	0	0	2	0	0
7	1	0	1	0	0	0
8	1	0	1	0	0	0
9	1	0	1	2	0	0
10	0	1	0	0	3	0
11	0	1	0	0	0	0
12	0	1	1	0	0	0
13	0	1	0	0	3	0
14	0	1	0	0	0	0
15	0	1	1	0	0	0
16	0	1	0	0	3	0
17	0	1	0	0	0	0
18	0	1	1	0	0	0
19	0	0	1	0	0	0
20	0	0	1	0	0	0
21	0	0	1	0	0	0
22	0	0	1	0	0	0
23	0	0	1	0	0	0
24	0	0	1	0	0	0
25	0	0	1	0	0	0
26	0	0	1	0	0	0
27	0	0	1	0	0	0
28	0	0	1	1	0	0
29	0	0	0	1	0	0
30	0	0	0	1	3	0
31	0	0	1	1	0	0
32	0	0	0	1	0	0
33	0	0	0	1	3	0
34	0	0	1	1	0	0
35	0	0	0	1	0	0
36	0	0	0	1	3	0
37	0	0	0	2	1	0
38	0	0	0	0	1	0
39	0	2	0	0	1	0
40	0	0	0	2	1	0
41	0	0	0	0	1	0
42	0	2	0	0	1	0
43	0	0	0	2	1	0
44	0	0	0	0	1	0
45	0	2	0	0	1	0
46	0	2	1	0	0	1
47	0	0	1	0	0	1
48	0	0	1	0	0	1
49	0	2	0	0	0	1
50	0	0	0	0	0	1
51	0	0	0	0	0	1
52	0	2	0	0	3	1
53	0	0	0	0	3	1
54	0	0	0	0	3	1

Table 5.1: Relative orientation change of each facelet for clockwise quarter turns of each face

However, we can consider sums of columns because

$$\begin{aligned}
& s_\delta(m) = s_\delta(\mathrm{turn}(m, Q)) \\
\iff & s_\delta(m) = s_\delta(m') \\
\iff & \left(\sum_{i \in \mathcal{I}} m_i\right) \bmod 2 = \left(\sum_{i \in \mathcal{I}} m'_i\right) \bmod 2 \\
\iff & \left(\sum_{i \in \mathcal{I}} m_i\right) + \left(\sum_{i \in \mathcal{I}} m'_i\right) \equiv 0 \pmod 2 \\
\iff & \left(\sum_{i \in \mathcal{I}} m_i\right) + \left(\sum_{i \in \mathcal{I}} m_i\right) + \left(\sum_{i \in \mathcal{I}} \mathbf{R}_{i,Q}\right) \equiv 0 \pmod 2 \\
\iff & \sum_{i \in \mathcal{I}} \mathbf{R}_{i,Q} \equiv 0 \pmod 2.
\end{aligned}$$

The last congruence can be verified using Table 5.1. Indeed, the sum of each column is even for $i \in \mathcal{I}$:

$$\sum_{i \in \mathcal{I}} \mathbf{R}_{i,U} = 8 \qquad \sum_{i \in \mathcal{I}} \mathbf{R}_{i,L} = 20 \qquad \sum_{i \in \mathcal{I}} \mathbf{R}_{i,F} = 20$$

$$\sum_{i \in \mathcal{I}} \mathbf{R}_{i,R} = 20 \qquad \sum_{i \in \mathcal{I}} \mathbf{R}_{i,B} = 44 \qquad \sum_{i \in \mathcal{I}} \mathbf{R}_{i,D} = 8$$

Therefore, $s_\delta(m) = s_\delta(\mathrm{turn}(m, Q))$ for any $m \in \mathcal{M}$ and $Q \in \{ U, L, F, R, B, D \}$. ☐

Corollary 5.5. *Let $m \in \mathcal{M}$ be a state and $q \in \{ U, L, F, R, B, D \}^*$ be a sequence of faces. Then $s_\delta(m) = s_\delta(\mathrm{turn}^*(m, q))$.*

Proof. This follows from Lemma 5.4 by induction.

Clearly, if the length ℓ of q is 0, then $\mathrm{turn}^*(m, q) = m$ and, thus, $s_\delta(\mathrm{turn}^*(m, q)) = s_\delta(m)$. Otherwise, the length ℓ is greater than zero. Let Q be the last turn in q and q' be q up to but not including Q. The length of q' is $\ell - 1$. Let $m' := \mathrm{turn}^*(m, q')$. Then, by inductive hypothesis, $s_\delta(m') = s_\delta(m)$. According to Lemma 5.4, we have $s_\delta(\mathrm{turn}^*(m, q)) = s_\delta(\mathrm{turn}(m', Q)) = s_\delta(m') = s_\delta(m)$. ☐

Corollary 5.6. *A state $m \in \mathcal{M}$ cannot, through a sequence of turns, be transformed into a state $m' \in \mathcal{M}$ if $s_\delta(m) \neq s_\delta(m')$.*

Proof. This follow immediately from Corollary 5.5. Any state m'' that can be reached from m fulfills $s_\delta(m'') = s_\delta(m)$, therefore, $m'' \neq m'$, which means that m' cannot be reached from m. □

Algorithm 5.1 IND-CPA adversary based on s_δ

```
Output: $b \in \{0, 1\}$
1: $m_1 \leftarrow$ plaintext with $s_\delta(m_1) = 0$                ▷ see Figure 5.2a
2: $m_2 \leftarrow$ plaintext with $s_\delta(m_2) = 1$                ▷ see Figure 5.2b
3: $(c, r) \leftarrow \mathrm{LR}(m_1, m_2)$          ▷ challenger encrypts either $m_1$ or $m_2$
4: if $s_\delta(c) = s_\delta(m_1)$ then
5:     return $0$                          ▷ challenger encrypted the left message
6: else
7:     return $1$                          ▷ challenger encrypted the right message
```

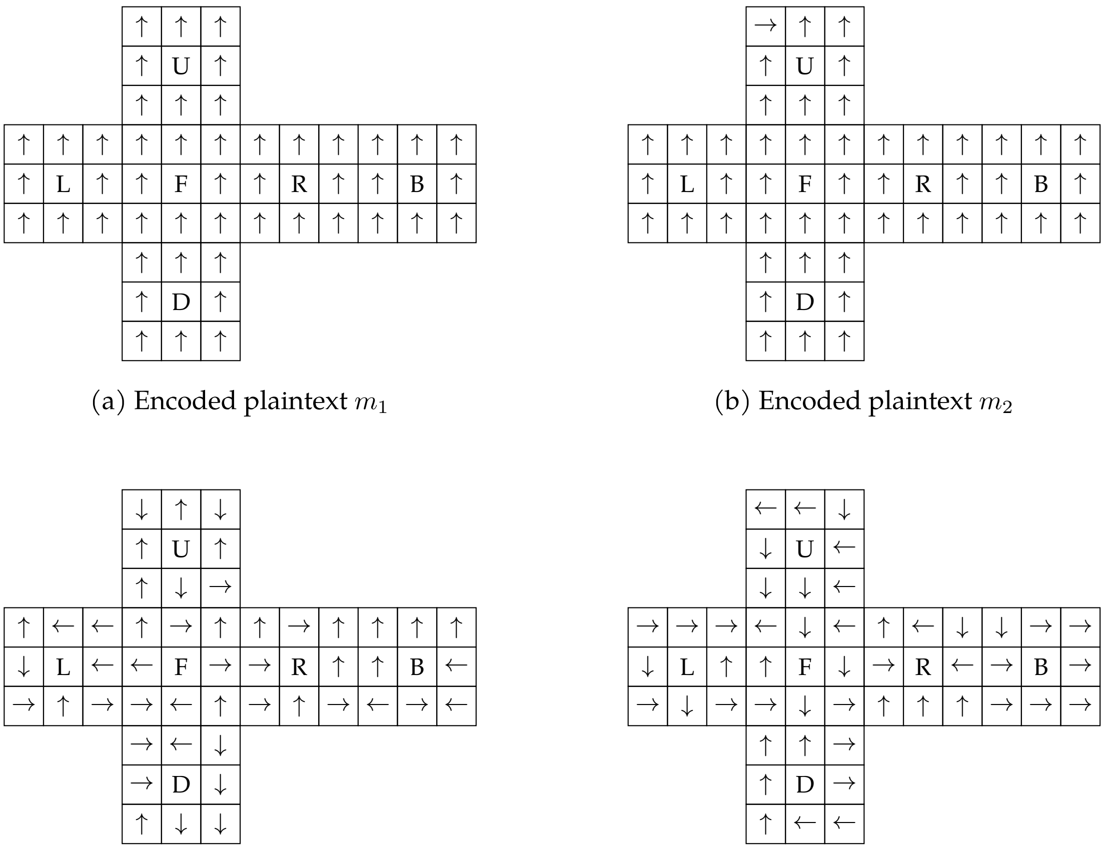
Theorem 5.7. *The symmetric encryption scheme $\mathfrak{S}_1$ lacks indistinguishability under chosen plaintext attack (IND-CPA).*

Proof. Let $m_1, m_2 \in \mathcal{M}$ such that $s_\delta(m_1) \neq s_\delta(m_2)$. Without loss of generality, let $s_\delta(m_1) = 0$ and $s_\delta(m_2) = 1$.

Let (c_1, r_1) and (c_2, r_2) be the outputs that are produced by the encryption algorithm (see Section 5.2) upon inputs m_1 and m_2, respectively. Without knowing the secret key, it is known that c_1 and c_2 were produced by applying sequences of turns to m_1 and m_2, respectively. Therefore, according to Corollary 5.5, we have $s_\delta(c_1) = s_\delta(m_1) = 0$ and $s_\delta(c_2) = s_\delta(m_2) = 1$.

Thus, an attacker can distinguish the ciphertexts based on s_δ and can relate them to the chosen plaintexts (see Algorithm 5.1), which means that the scheme does not provide indistinguishability under chosen plaintext attack. □

Remark 5.8. Note that s_δ uses $\mathcal{I}$, which does not include center facelet indices. Therefore, even when removing the center facelets from the scheme in response to the attack described in Section 5.3, this attack succeeds.



(a) Encoded plaintext m_1

(b) Encoded plaintext m_2

(c) Encoded ciphertext c_1

(d) Encoded ciphertext c_2

Figure 5.2: Illustration of a chosen-plaintext attack against $\mathfrak{S}_1$ using s_δ

This attack is demonstrated in Figure 5.2. Since all arrows in Figure 5.2a point up, we have $s_\delta(m_1) = 0$. Conversely, Figure 5.2b shows m_2, in which all arrows but one point up, and the sole arrow not pointing up is perpendicular to $\uparrow$, therefore, $s_\delta(m_2) = 1$. The ciphertext c_1, visualized in Figure 5.2c, fulfills, as predicted by Corollary 5.5, $s_\delta(c_1) = 0$. According to Corollary 5.6, c_1 could not possibly be the result of applying a sequence of turns to m_2. Conversely, $s_\delta(c_2) = 1$ and, therefore, c_2 cannot be the result of encrypting m_1.

Remark 5.9. Pan et al. [48] attempt to prove IND-CPA security by relating the problem of distinguishing ciphertexts to the conjugacy decision problem (see Definition 3.3) for the Rubik's cube group $\mathfrak{R}_3$. However, as shown here, it is possible to break IND-CPA security without considering k and r (see Section 5.2) at all, which play an essential role in their proof. They claim that the existence of an IND-CPA attacker implies the existence of a probabilistic conjugacy problem solver, however, as part of the proof, they present the IND-CPA attacker with an input that could not have been created by the encryption algorithm, thus violating the IND-CPA protocol.

Indistinguishability under chosen plaintext attack (IND-CPA) is a fundamentally important property of encryption schemes. Pan et al. [48] present a variant of $\mathfrak{S}_1$ that uses a well-known construction that was proposed by Fujisaki and Okamoto in 1999 [20] to achieve indistinguishability under adaptive chosen ciphertext attacks (IND-CCA2). However, with IND-CPA security of $\mathfrak{S}_1$ disproven, the IND-CCA2 security property of the modified scheme $\mathfrak{S}_2$ that was claimed by Pan et al. [48] has to be discarded as well.

Corollary 5.10. *The symmetric encryption scheme $\mathfrak{S}_2$ lacks indistinguishability under adaptive chosen ciphertext attack (IND-CCA2).*

Proof. The scheme $\mathfrak{S}_2$ lacks IND-CPA security since it is based on $\mathfrak{S}_1$ and the differences between these two schemes do not affect the previously described attack. The output of the encryption algorithm associated with $\mathfrak{S}_2$ still allows distinguishing between ciphertexts based on s_δ. $\square$

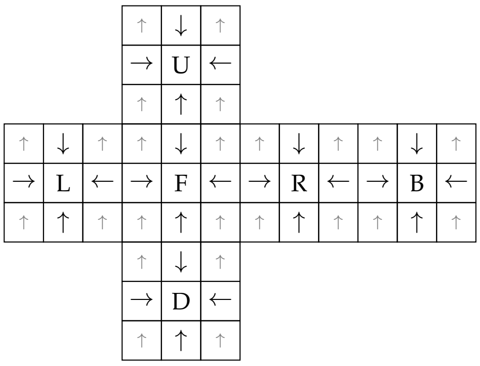
Remark 5.11. Observant readers might consider a modification of the proposed scheme that not only excludes the centers from the model (see Section 5.3), but also reduces the block size by another bit, whose value is then determined by the encryption algorithm based on the rest of the message such that s_δ is constant for all messages. However, the attack described in this section also works with definitions of s_δ that use certain subsets of $\mathcal{I}$ instead of $\mathcal{I}$, and such a modification would fail against at least one of these alternative definitions.

5.5 Attacks based on invariant structural properties

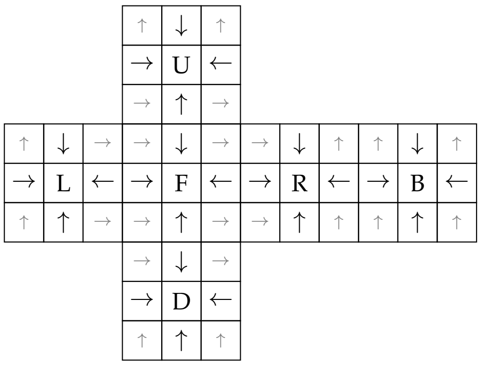
This section briefly discusses another class of attacks, which rely on invariant structural properties of the model used by Pan et al. [48]. These attacks are intuitively simpler than the previously described attack, but not immediately apparent in the formal model.

One such invariant is the orientation of the arrows on edge facelets relative to the edge that the facelet is connected to. In other words, the orientation and position of these arrows may change, but each arrow will maintain the same angle to the nearest edge, regardless of its position or orientation. In most cases, given the ciphertext only, this invariant does not allow conclusions about the original message because the original positions and orientations of the edge facelets are still unknown. However, when an attacker is able to choose a plaintext, for example, in the IND-CPA security model, it becomes trivial to distinguish ciphertexts based on such invariants.

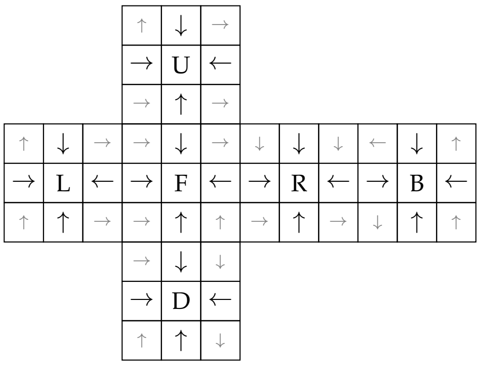
Figure 5.3 demonstrates one such possibility: when all arrows on all edge facelets point toward the centers of their respective faces, then this will be true for any state that results from turning faces of the Rubik's cube. The position and orientation of the facelets changes, but, since the model ignores the colors of the facelets, this has no effect.



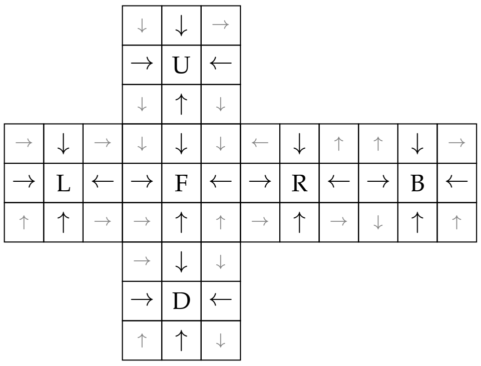
(a) Initial state



(b) After turning F

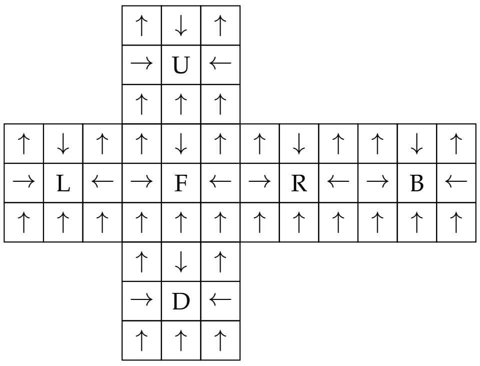


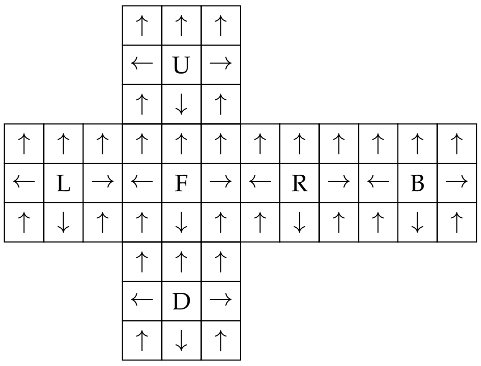
(c) After turning F and R

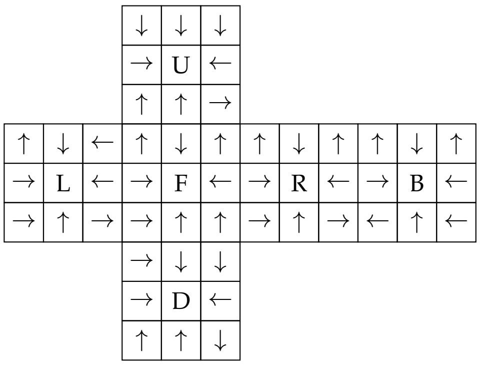


(d) After turning F, R, and U

Figure 5.3: Relative orientation of edge facelets as an invariant


(a) Encoded plaintext m_1


(b) Encoded plaintext m_2


(c) Encoded ciphertext c_1

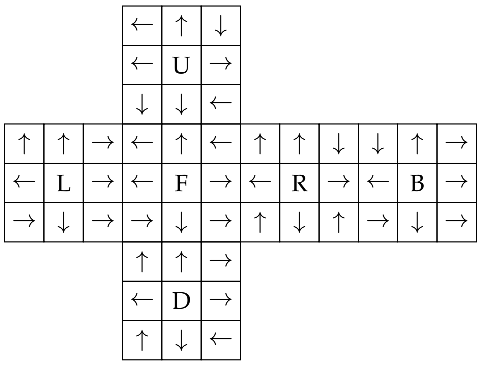

(d) Encoded ciphertext c_2

Figure 5.4: Illustration of a chosen-plaintext attack against $\mathfrak{S}_1$ using edge facelets

Figure 5.4 shows how such invariants can be used to disprove IND-CPA security in a less formal and more intuitive way than previously shown. The plaintext messages (see Figure 5.4a and Figure 5.4b, respectively) are

$$\begin{aligned} m_1 &:= \texttt{001000010011000000001000010011000000} \\ &\ \texttt{001000010011000000001000010011000000} \\ &\ \texttt{001000010011000000001000010011000000}, \\ m_2 &:= \texttt{000000110001001000000000110001001000} \\ &\ \texttt{000000110001001000000000110001001000} \\ &\ \texttt{000000110001001000000000110001001000}. \end{aligned}$$

The secret key is $k :=$ FDRDLUBU and the randomly chosen turn sequences are

$$\begin{aligned} r_1 &:= \text{RBUFRRUL} \quad \text{(for encrypting } m_1\text{)}, \\ r_2 &:= \text{ULLDRUUB} \quad \text{(for encrypting } m_2\text{)}. \end{aligned}$$

The ciphertexts (see Figure 5.4c and Figure 5.4d, respectively) are

$$\begin{aligned} c_1 &= \texttt{1010100110110000001001011010111010001} \\ &\ \texttt{0010000101110100000001000011111010001} \\ &\ \texttt{001000010111110011011010010011000010}, \\ c_2 &= \texttt{1100101111011010110000001111001011001} \\ &\ \texttt{110011110001011001000010110101001000} \\ &\ \texttt{1000011101010110010000001110101001011}. \end{aligned}$$

Because the arrows on the edge facelets are indistinguishable since they all have the same orientation relative to their nearest edge, their positions and absolute orientations are irrelevant. It is trivial for an attacker in the IND-CPA model to distinguish between ciphertexts by only considering the orientation of the arrows on the edge facelets if said attacker chooses the plaintexts accordingly, for example, as shown in Figures 5.4a and 5.4b. Thus, as already formally proven in the previous section, the scheme $\mathfrak{S}_1$ is not IND-CPA secure.

5.6 Key recovery attacks

None of the previously described attacks allow recovering the secret key k. While they are sufficient to disprove IND-CPA and IND-CCA security (see Section 5.4 and Section 5.5) and even allow recovering parts of the plaintext (see Section 5.3), they do not allow an attacker to fully decrypt arbitrary messages nor do they allow an attacker to encrypt messages in a manner that would lead to a desired plaintext upon decryption by a legitimate receiver. Recovering the key k in the scheme $\mathfrak{S}_1$ appears to be difficult for two reasons:

1. Knowing the ciphertext does not imply knowing the Rubik's cube's state that resulted from the turn sequence that was applied by the encryption algorithm.
2. Even if the Rubik's cube's state was known to an attacker, the conjugacy search problem (see Definition 3.6) is presumably difficult for the Rubik's cube group, meaning that recovering k from the cube's state is assumed to be infeasible [48].

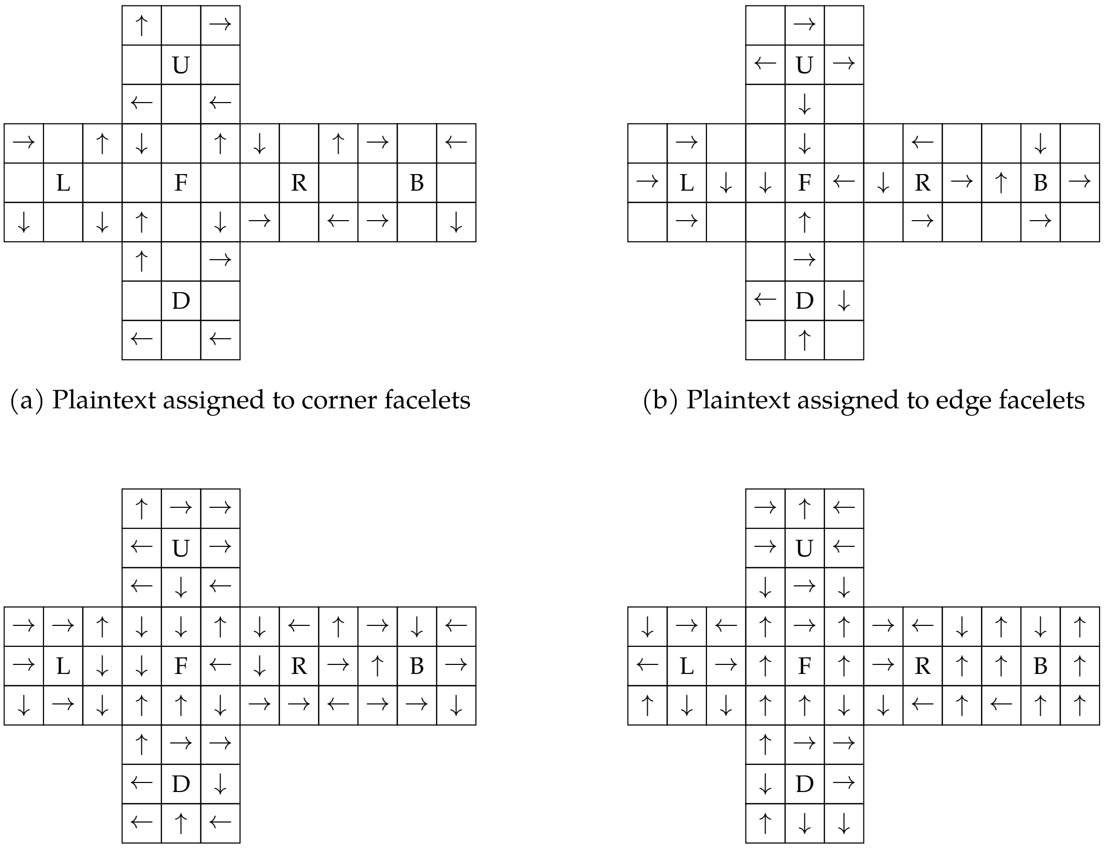
While the first argument might be true in the general case, it is not when the attacker can choose a plaintext.

5.6.1 Obtaining the Rubik's cube's state

Definition 5.12. A *multiset* S of cardinality $|S|$ is an unordered collection of $|S|$ not necessarily distinct elements.

Lemma 5.13. *The number of different multisets that only contain elements from a set S and whose cardinality is precisely z is*

$$\binom{z + |S| - 1}{z}.$$


(a) Plaintext assigned to corner facelets

(b) Plaintext assigned to edge facelets

(c) Crafted plaintext

(d) Resulting ciphertext

Figure 5.5: Crafted plaintext and resulting ciphertext for recovering the Rubik's cube's state that is used by the encryption algorithm

There are eight corner cubies, each having three facelets. The orientations of the arrows on the facelets on a corner cubie, relative to the corner, does not change when applying turns to the Rubik's cube. This is similar to the invariant relative orientation of the arrows on edge facelets relative to their nearest edge as described in Section 5.5. The position and orientation of these arrows does change with the position and orientation of the cubie that they are on. Still, regardless of any turns applied to the cube, the multiset of orientations of the arrows on each corner cubie, relative to the corner, remains the same regardless of the position and orientation of the cubie. The cardinality of the multiset is three and its elements must be from the set $\mathbb{Z}/4\mathbb{Z} = \{0, 1, 2, 3\}$ and there are 20 such multisets. Because there are more such multisets than corner cubies, the orientations of the arrows on the corner cubies can be chosen such that, even after applying an arbitrary sequence of turns to the Rubik's cube, corner cubies can be identified uniquely using the relative orientations of the arrows on the three facelets that belong to the cubie.
Each center cubie only has exactly one facelet and the position of those cubies does not change when applying turns to the Rubik's cube (see Section 5.3).
The remaining cubies are edge cubies. There are twelve edge cubies, each having two facelets. Section 5.5 already discussed the fact that the orientations of the arrows on edge facelets, relative to the nearest edge, do not change. However, there are only 10 multisets, therefore, it is impossible to uniquely identify edge cubies based on such multisets alone. Further, to determine the original position of each facelet, it is insufficient to obtain the original position of each cubie because the orientation of the cubie may change as well, and multisets that only contain one distinct element are insufficient for determining the orientation of a cubie. Therefore, it makes sense to only assign multisets that contain at least two distinct elements. There are 16 such multisets with cardinality 3, which means that there are enough such multisets to uniquely identify each corner cubie (see Figure 5.5a. However, there are only 6 such multisets with cardinality 2. Assigning the same multiset to two edge cubies means that, for each such pair, it is not possible to distinguish between the two edge cubies based on arrow orientations alone (see Figure 5.5b).

By choosing a plaintext that assigns such multisets to edge and corner cubies, the attacker can draw conclusions about the Rubik's cube state from the ciphertext. The position and orientation of corner cubies can be determined unambiguously whereas the position of the edge cubies cannot. However, for each of the six pairs of edge cubies, only two possible positions exist. Therefore, the Rubik's cube must be in one of $2^6 = 64$ possible states. Compared to the total number of possible states of the Rubik's cube, which is approximately 1.2×2^{65}, this attack accomplishes a significant improvement, and is perhaps the most interesting result within this chapter.

Example. Let $m \in \mathcal{M}$ be the crafted plaintext from Figure 5.5c. The bit sequence that represents m is

$$\begin{aligned} m := {} & \texttt{000101110001111011010100010010100110} \\ & \texttt{101000100011000010101100100001010111} \\ & \texttt{011011000001010110000101110010110011}. \end{aligned}$$

We encrypt m using the secret key k := RRFLUDRDLUBU and the randomly chosen turn sequence r := BULLBDRUUL. The resulting ciphertext (see Figure 5.5d) is

$$\begin{aligned} c = {} & \texttt{010011011111100110100111111101001010} \\ & \texttt{000100000000000010011110010100101100} \\ & \texttt{001000001000110000000101100101001010}. \end{aligned}$$

We then proceed as described, without using k, to obtain the 64 possible states of the Rubik's cube, which are shown in Figure 5.6. The highlighted state is indeed the correct state (as can be verified by turning a Rubik's cube according to $k^{-1}rk$).

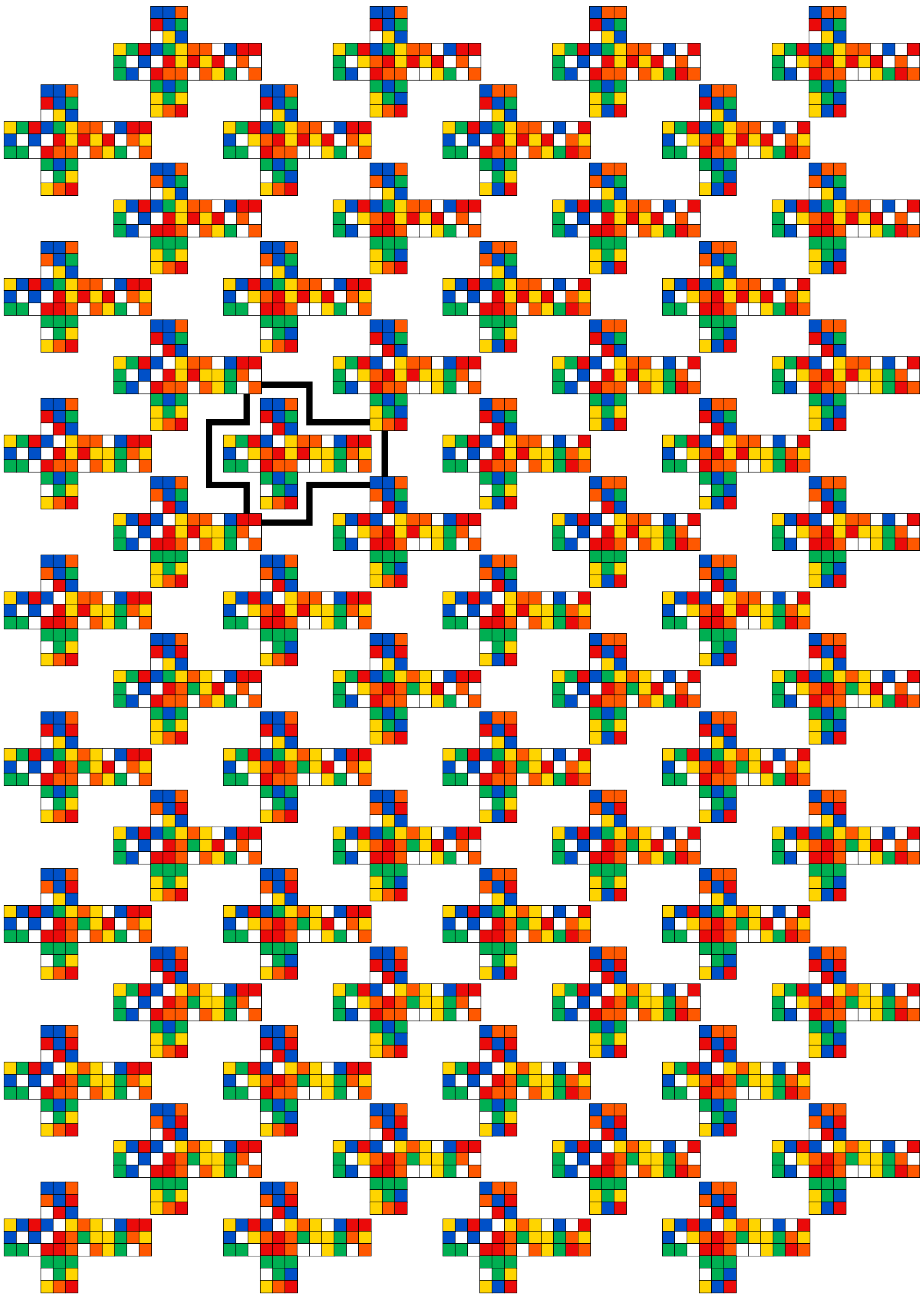


Figure 5.6: Rubik's cube states recovered from ciphertext. The correct state is highlighted.

Theorem 5.14. *Knowing the Rubik's cube state $\sigma = k^{-1}rk$ that was used to produce a ciphertext is sufficient for producing arbitrary ciphertexts that, upon decryption, result in chosen plaintexts.*

Proof. (Sketch only.) While it is (presumably) difficult to obtain k from σ even when r is known, we can statically determine the position (directly from σ) and orientation change (refer to Table 5.1) of each facelet incurred by σ (e.g., by solving σ through an arbitrarily long sequence of turns, which is efficiently computable). For the orientation change of the center facelets, only r has to be considered. Then, proceed as in the encryption algorithm (with r fixed) for a chosen plaintext $m \in \mathcal{M}$ (see Section 5.2). The obtained ciphertext $(m', r) \in \mathcal{M} \times \mathfrak{R}_3$ will decrypt to m when the receiver uses the key k because $\sigma k^{-1}r^{-1}k = (k^{-1}rk)k^{-1}r^{-1}k = k^{-1}rr^{-1}k = k^{-1}k = \mathrm{id}_{\mathfrak{R}_3}$. □

Corollary 5.15. *If an attacker can encrypt a single chosen plaintext, they can forge any number of messages with success probability $\frac{1}{64}$.*

Proof. Proceed as described above to obtain $(m', r) \in \mathcal{M} \times \mathfrak{R}_3$ and the 64 possible states of the Rubik's cube. Then proceed as in Theorem 5.14 by guessing which of the 64 states was used for encryption. □

Remark 5.16. The scheme $\mathfrak{S}_2$, for which Pan et al. claimed IND-CCA2 security [48], is *at least* as vulnerable as the scheme $\mathfrak{S}_1$ that is being discussed here. The attack described in Corollary 5.15 works against $\mathfrak{S}_2$ as well because the attacker can determine the hash value $H(m, r)$ for any chosen plaintext $m \in \mathcal{M}$ and known $r \in \mathfrak{R}_3$. Additionally, because $\mathfrak{S}_2$ uses the same sequence r to encrypt $H(m, r)$, the attacker effectively obtains two ciphertexts that use the same key k and the same rotating sequence r (and thus the same Rubik's cube state $\sigma = k^{-1}rk$) in the first step of the attack. The ciphertext that results from encrypting $H(m, r)$ can likely be used to reduce the set of possible states of the Rubik's cube down further, which increases the success probability of the attack beyond $\frac{1}{64}$.

However, knowing a single Rubik's cube state is insufficient for decrypting other ciphertexts since the Rubik's cube's state depends on the turn sequence r that is chosen by the encryption algorithm.

5.6.2 Solving the conjugacy search problem

As the previous section demonstrated, it is possible to recover the Rubik's cube's state that is used by the encryption algorithm. However, recovering the key k from the state is still difficult.

When represented as permutations over $\mathfrak{R}_3$, the recovered state σ fulfills $\sigma = k^{-1}rk$. Obtaining k requires a solution to the conjugacy search problem (see Definition 3.6) for the Rubik's cube group $\mathfrak{R}_3$, which is presumed to be difficult (see Section 3.1).

Nevertheless, even an exhaustive search for k only needs to search the key space $\mathfrak{R}_3$, which contains 1.2×2^{65} possible keys. While this might be impractical, such a small key space is generally considered insecure by today's standards [4].

Quantum computing attacks, e.g., using Grover's algorithm, can reduce the number of operations to $\mathcal{O}\left(\sqrt{N}\right)$, where N is the order of the group [25]. In this case, a quantum computer could find k from σ in approximately 2^{33} iterations.

5.7 Analysis of discovered weaknesses

All previously presented weaknesses and attacks result from the fact that all operations that act on the Rubik's cube preserve certain properties.

- The sets of edge facelets, corner facelets, and center facelets are disjoint and do not change. In particular, the positions of center facelets do not change (see Section 5.3).
- Some congruences are preserved (see Section 5.4).
- The orientation of edge facelets, relative to their nearest edge, does not change (see Section 5.5).
- The orientation of corner facelets, relative to their nearest corner, does not change (see Section 5.6.1).

Another way of looking at these issues is as follows. The message space and ciphertext space are $\{0,1\}^{108}$ each. Fix a plaintext message. Then the arrow orientations on the corner facelets and edge facelets depend on the permutation $k^{-1}rk \in \mathfrak{R}_3$ only. The arrow

Block cipher	**Block size**	**Key size**
IDEA (1991) [37]	64 bits	128 bits
Triple DES (1995) [29]	64 bits	112 bits or 168 bits
Twofish (1998) [57]	128 bits	128 bits or 192 bits or 256 bits
Rijndael (1998) [11]	128 bits	128 bits or 192 bits or 256 bits
Camellia (2000) [3]	128 bits	128 bits or 192 bits or 256 bits
$\mathfrak{S}_1$, $\mathfrak{S}_2$ (2020) [48]	108 bits	65 bits (approx.)

Table 5.2: Comparison of the block size and key size of the scheme due to Pan et al. [48] to other block ciphers

orientations on the center facelets do not depend on k but only on r and are always in one of only 4^6 possible states since there are six such facelets and each has two bits assigned to it. Combined, this means that, for the fixed plaintext message, regardless of the choice of k and r, the ciphertext must be one of $|\mathfrak{R}_3| \cdot 4^6 = 1.2 \times 2^{77}$ possible bit sequences.
The ciphertext space is 1.7×2^{30} times larger than the number of possible ciphertexts for a fixed plaintext. The attacks in Section 5.4 and in Section 5.5 took advantage of this fact by choosing plaintexts such that the sets of possible ciphertexts were disjoint and such that the resulting ciphertexts could be attributed to the respective sets of possible ciphertexts.
As Table 5.2 shows, the block cipher $\mathfrak{S}_1$ proposed by Pan et al. [48] uses a much smaller key space than commonly used block ciphers, and it is unusual for a modern block cipher to use block sizes that are larger than the key size, measured in bits. In this case, it even leads to catastrophic security flaws of the proposed scheme. Conversely, reducing the block size could result in an unusually small block size compared to modern block ciphers.
The evaluation performed by Pan et al. [48] focuses on performance. While the required computations might be fast compared to other encryption schemes (however, refer to Section 5.10.2 for reasons as to why the performance evaluation should be disregarded), the analysis ignores the fact that the proposed scheme adds a randomly chosen turn sequence r (or, equivalently, a randomly chosen Rubik's cube state) to each encrypted block. The publication uses a sequence of ten turns for r in an example, which can be encoded as a

sequence of $\lceil \log_2(6^{10}) \rceil = 26$ bits. This increases the effective size of the ciphertext from 108 bits to 134 bits, which is an increase by 24 %. Unlike other symmetric cryptosystems that add a random nonce per message, the proposed scheme adds this random sequence to each block. As a result, encrypting longer messages will result in a significant overhead. It is worth mentioning that the block size is not a multiple of eight bits. This is unusual for a modern block cipher. Because application data usually consists of units of eight bits, this aspect potentially complicates usage. However, when not assigning data bits to the center facelets of the Rubik's cube due to the weakness discussed in Section 5.3, the block size is reduced to 96 bits, which is a multiple of eight bits.

Similarly, the key k and the rotation sequence r can either be encoded as elements of $\mathfrak{R}_3$ or as sequences of clockwise quarter turns of the six faces. In the first case, the set of possible values is $\mathfrak{R}_3$, and the order of $\mathfrak{R}_3$ is 1.2×2^{65}, therefore, no bijective mapping from bit sequences of fixed length to $\mathfrak{R}_3$ exists: there are not enough bit sequences of length 65 and too many bit sequences of length 66. When encoding the values as turn sequences, for a sequence length ℓ, there are 6^ℓ different turn sequences. However, there are no integer solutions for b in $2^b = 6^\ell$ for $\ell \in \mathbb{N}$, which means that again, no bijective mapping from bit sequences of fixed length to turn sequences of fixed length exists. In other words, no bijective mapping from bit sequences of fixed length to values for k and r exists, which is another potential usability issue.

Table 5.2 includes Triple DES [29], which is based on the weaker DES block cipher. It is possible to use a similar construction to create a "Triple $\mathfrak{S}_1$" block cipher with the primary goal of increasing the key size beyond 100 bits. However, such a construction would not be sufficient to address the fundamental weaknesses of the proposed scheme.

Similarly, using larger Rubik's cubes would increase the group order and thus the key space as well as the block size. However, just like the the standard 3x3x3 Rubik's cube, larger Rubik's cubes possess structural properties that preserve certain patterns in the message regardless of their size, e.g., regardless of the size of the Rubik's cube, corner cubies will always remain in a corner position.

5.8 Potential modification to address discovered weaknesses

At its core, the scheme proposed by Pan et al. [48] applies the face turn sequences k^{-1}, r, and k to a Rubik's cube whose facelet orientations encode the message, where k is the secret symmetric key and r is a randomly chosen turn sequence. While this idea is interesting and might seem promising at first, this chapter has revealed multiple weaknesses of the proposed scheme that effectively break its security almost entirely.
As discussed in Section 5.7, the weaknesses arise from structural properties of the Rubik's cube and from the small key space.
A potential workaround could be an additional operation τ that permutes the positions of the facelets in an *unnatural* manner, i.e., in a pattern that cannot be produced by turns of the faces of the cube.
Such a permutation τ could, for example, shift all facelets upward (in the two-dimensional model of the Rubik's cube), placing the top-most facelets in the bottom-most positions.

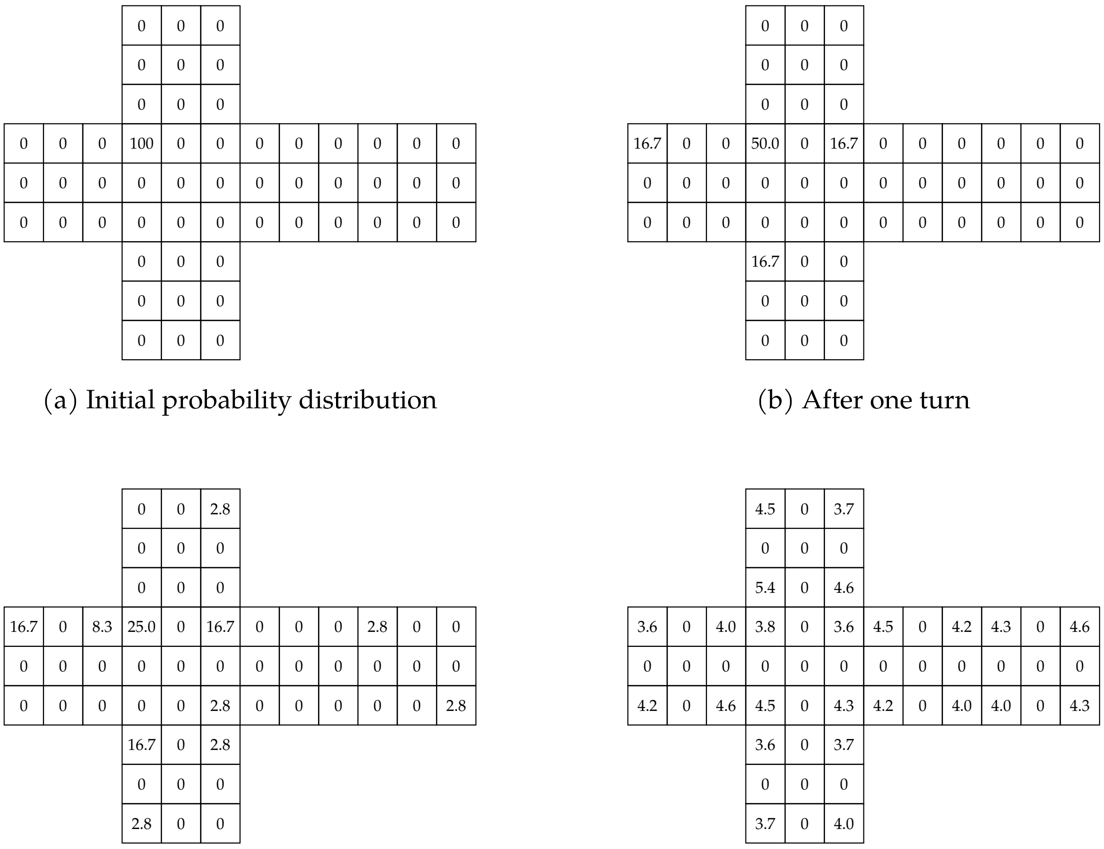
$$\begin{aligned}\tau := &(1\ 52\ 49\ 46\ 25\ 22\ 19\ 7\ 4)\\ &(2\ 53\ 50\ 47\ 26\ 23\ 20\ 8\ 5)\\ &(3\ 54\ 51\ 48\ 27\ 24\ 21\ 9\ 6)\\ &(10\ 16\ 13)\,(11\ 17\ 14)\,(12\ 18\ 15)\\ &(28\ 34\ 31)\,(29\ 35\ 32)\,(30\ 36\ 33)\\ &(37\ 43\ 40)\,(38\ 44\ 41)\,(39\ 45\ 42)\end{aligned}$$

This permutation moves • all six center facelets to the positions of edge facelets, • those six edge facelets to the positions of other edge facelets, • those six edge facelets to the positions of the center facelets, • the remaining twelve edge facelets to the positions of corner facelets, • those twelve corner facelets to the positions of the other twelve corner facelets, and • lastly, those twelve corner facelets to the positions of edge facelets.

Remark 5.17. If one wishes to model the Rubik's cube without center facelets, a different permutation τ must be chosen.

Such an operation τ could be a part of the key k and/or the random sequence r. Alternatively, it could be applied implicitly by the encryption algorithm between turns in the sequence $k^{-1}rk$, and τ^{-1} would then be applied by the decryption algorithm between turns in the sequence $k^{-1}r^{-1}k$.
The *unnatural* character of τ arises from the fact that it takes away any structural properties that would otherwise be preserved by the Rubik's cube. Through a sequence consisting of face turns and applications of τ, any facelet can move to any position.

Example. Figure 5.7 and Figure 5.8 show the probability distributions of the possible positions of a single corner facelet in the model used by Pan et al. [48] and in a modified model that applies τ between turns, respectively. In the original model, after two turns, the facelet must be in one of only 11 positions (see Figure 5.7c), whereas it can already be in one of 12 positions in the modified model after the same number of turns (see Figure 5.8c). In both models, after ten turns, the probability distribution is reasonably close to a uniform distribution. However, in the original model, that only applies to the set of positions of corner facelets (see Figure 5.7d). In the modified model, the probabilities of all 54 facelet positions approximate $1/54$ each (see Figure 5.8d).



			0	0	0						
			0	0	0						
			0	0	0						
0	0	0	100	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
			0	0	0						
			0	0	0						
			0	0	0						

(a) Initial probability distribution

			0	0	0						
			0	0	0						
			0	0	0						
16.7	0	0	50.0	0	16.7	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
			16.7	0	0						
			0	0	0						
			0	0	0						

(b) After one turn

			0	0	2.8						
			0	0	0						
			0	0	0						
16.7	0	8.3	25.0	0	16.7	0	0	0	2.8	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2.8	0	0	0	0	0	2.8
			16.7	0	2.8						
			0	0	0						
			2.8	0	0						

(c) After two turns

			4.5	0	3.7						
			0	0	0						
			5.4	0	4.6						
3.6	0	4.0	3.8	0	3.6	4.5	0	4.2	4.3	0	4.6
0	0	0	0	0	0	0	0	0	0	0	0
4.2	0	4.6	4.5	0	4.3	4.2	0	4.0	4.0	0	4.3
			3.6	0	3.7						
			0	0	0						
			3.7	0	4.0						

(d) After ten turns

Figure 5.7: Probability of positions of the top-left facelet of the front face (F) in the model used by Pan et al. [48] (as rounded percentages)

			0	0	0						
			0	0	0						
			0	0	0						
0	0	0	100	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
			0	0	0						
			0	0	0						
			0	0	0						

(a) Initial probability distribution

			0	0	0						
			0	0	0						
			0	0	0						
16.7	0	0	50.0	0	16.7	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
			16.7	0	0						
			0	0	0						
			0	0	0						

(b) After one turn

			8.3	0	0						
			0	0	0						
			27.8	0	8.3						
2.8	0	0	2.8	0	0	8.3	0	0	2.8	0	0
0	0	0	0	0	0	0	0	0	0	0	0
8.3	0	0	19.4	0	0	5.6	0	0	0	0	0
			0	0	0						
			0	0	0						
			2.8	0	2.8						

(c) After two turns

			2.4	1.6	2.4						
			2.4	1.8	1.5						
			1.8	2.5	1.8						
2.4	1.1	2.3	3.0	1.6	2.7	1.9	1.4	1.9	1.7	1.2	1.8
1.7	1.2	1.4	2.5	1.9	1.5	2.5	1.3	1.9	1.7	0.9	1.8
1.9	1.6	1.8	1.8	2.3	1.8	1.7	1.9	1.5	1.8	1.3	1.9
			2.7	1.6	2.2						
			2.2	1.4	1.5						
			1.8	2.0	1.6						

(d) After ten turns

Figure 5.8: Probability of positions of the top-left facelet of the front face (F) in the modified model (as rounded percentages)

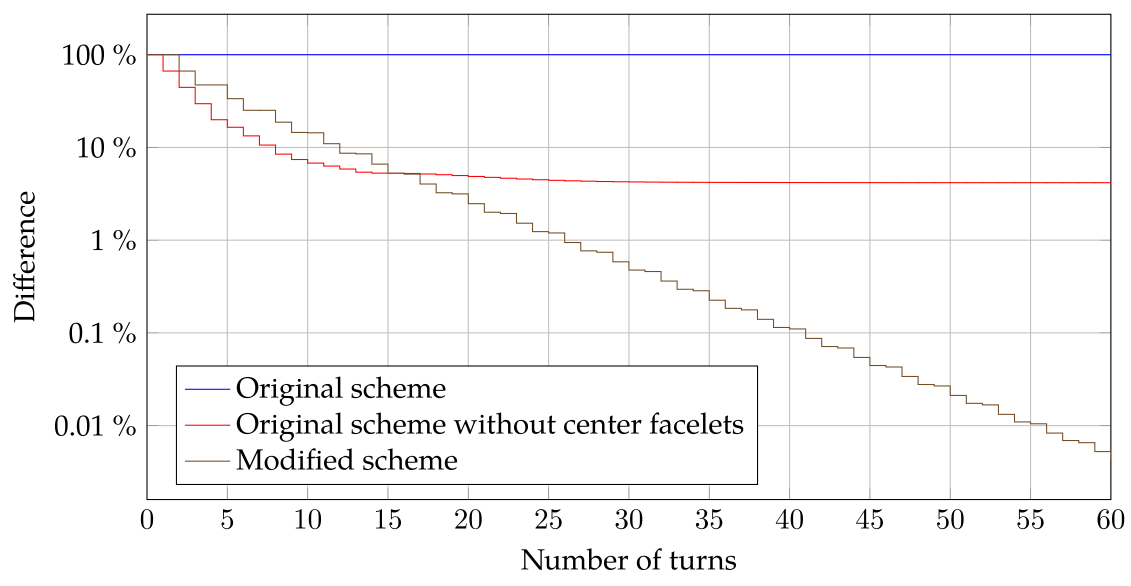


Figure 5.9: Difference between the probability of the most likely facelet position change and the probability of the least likely facelet position change, depending on the number of turns: $\max_{i,j}\{\,p_{i,j}\,\} - \min_{i,j}\{\,p_{i,j}\,\}$, where $p_{i,j}$ is the probability that the facelet that was originally in position $i \in \{\,1,\ldots,54\,\}$ is currently in position $j \in \{\,1,\ldots,54\,\}$.

As Figure 5.9 shows, in the modified scheme, the position of each facelet approaches a uniform probability distribution as the number of turns in the turn sequence increases.

These calculations only concern the position of each facelet. It could be that each facelet can be moved to any position, but not without restricting the set of possible positions of other facelets. To answer this question, let $\mathfrak{U}$, $\mathfrak{L}$, $\mathfrak{F}$, $\mathfrak{R}$, $\mathfrak{B}$, $\mathfrak{D}$ be the extensions of the Rubik's cube group generators U, L, F, R, B, and D to the model that includes the center facelets, respectively. The domain of these permutations is, therefore, $\{\,1,\ldots,54\,\}$, whereas the domain of the standard generators is $\{\,1,\ldots,48\,\}$ only.

Of course,

$$|\langle \mathfrak{U}, \mathfrak{L}, \mathfrak{F}, \mathfrak{R}, \mathfrak{B}, \mathfrak{D} \rangle| = |\langle U, L, F, R, B, D \rangle| = |\mathfrak{R}_3| \ll |\mathcal{S}_{54}|$$

holds. However, when applying τ before each of the turns,[2] we see that

$$|\langle \tau\mathfrak{U}, \tau\mathfrak{L}, \tau\mathfrak{F}, \tau\mathfrak{R}, \tau\mathfrak{B}, \tau\mathfrak{D} \rangle| = |\mathcal{S}_{54}|\,.$$

[2]It is irrelevant for this calculation whether τ is applied before the first turn or not.

Therefore, every permutation $\sigma \in S_{54}$ is indeed a possible reordering of the arrows on the facelets of the Rubik's cube in the modified model. From this result, it is also apparent that the key space must be at least as large as $\mathcal{S}_{54}$ because there must be at least one turn sequence producing each possible permutation of the facelets.

It is not necessarily obvious why this modified model is different from simply applying a permutation from $\mathcal{S}_{54}$ to a bit sequence, or why the key space is not necessarily exactly as large as $\mathcal{S}_{54}$. However, a permutation alone would hardly provide any level of security. Recall that, with each turn, not only does the position of the facelets on the surface of the Rubik's cube change, but also the orientations of the arrows on those facelets. In the original model, with the exception of the center facelets, the orientation change of each arrow only depends on its position change due to structural properties of the Rubik's cube. This is not true in the modified scheme.

Each of the 54 arrows on the facelets of the cube can point in one of four directions, therefore, the relative orientation change incurred by a sequence of turns, ignoring facelet positions, must be one of 4^{54}. Thus, there must be at least $|\mathcal{S}_{54}|$ and at most $|\mathcal{S}_{54}| \cdot 4^{54}$ non-equivalent turn sequences, i.e., keys.

While this modification appears to prevent the attacks presented in Section 5.3, Section 5.5, and Section 5.6, it does not result in IND-CPA security by itself. Attacks based on the idea that has been discussed in Section 5.4 also work against the modified scheme.

5.9 Valuation according to Shannon

In 1945, only a few years after Alan Turing proposed the idea of modern computers in the form of universal Turing machines [69], Claude Shannon developed theoretical foundations of cryptography and cryptanalysis [60]. Because his work was deemed critical to the national security of the country he was residing in, a redacted version was published four years later [61], and the original paper was not made publicly available for more than six decades. Despite the scientific advances in the fields of cryptography and cryptanalysis, his work has not lost its relevance, even though, at the time of writing, only symmetric cryptography had been developed. Modern cryptography, including cryptographic hash functions, asymmetric cryptography, digital signatures, and zero-knowledge protocols, had not been invented yet.

5.9.1 Criteria

According to Shannon [61, pp. 669-670], the five most important criteria for evaluating a symmetric cipher are 1. the amount of secrecy, 2. the size of the key, 3. the complexity of encryption and decryption, 4. the propagation of errors, and 5. message expansion (i.e., overhead due to ciphertexts being longer than plaintexts).

The *amount of secrecy* of the block cipher proposed by Pan et al. [48] has been thoroughly proven to be marginal in the previous sections.

A brute-force search for the key in any cryptosystem refers to an exhaustive search for the key in the entire key space. Therefore, the computational complexity of such a search is linear in the size of the key space.[3] Therefore, the *size of the key* is an important upper bound on the security of a cipher. Conversely, if the security of the cipher can be compromised through a significantly smaller computational effort than a brute-force search for the key, then the key space and, therefore, the key size has little to no advantage over a smaller key space and key size. In the scheme proposed by Pan et al. [48], the key size is too small to provide a sufficient level of security (see Section 5.6.2) and, at the same time, unnecessarily large when compared to the overall security of the scheme.

The *complexity of the encryption and decryption algorithms* is low. The computational complexity is at most linear in the length of the turn sequences. However, unlike most modern cryptosystems, such as Twofish [57] and Rijndael [11], the algorithmic design was not explicitly optimized for modern computer architectures and processors. It seems unlikely that a much simpler and much faster algorithm, such as the one proposed by Pan et al. [48], could achieve a level of security that is comparable to these modern ciphers.

[3]This is not true when considering quantum computing. On a sufficiently advanced quantum computer, the computational complexity is linear in the square root of the size of the key space [25].

The *propagation of errors* refers to the effects that a small change in the ciphertext has on the decrypted message. Shannon argued that such errors in the ciphertext should ideally result in few or localized errors [61]. This appears desirable when transmitting encrypted messages through an analog channel, however, with the shift toward digital data transmissions, multilayer networks, and forward error correction, transmission errors are unlikely to occur in application data. Additionally, it has become common practice to use authenticated encryption algorithms or message authentication algorithms prior to decoding received messages to ensure the messages' authenticity. These mechanisms intentionally prevent decoding modified messages. Therefore, this criterion does not seem as relevant today as it may have in 1945. Nevertheless, many modern encryption schemes do minimize such errors, for example, the commonly used counter mode of Rijndael (AES-CTR) [18]. In general, due to the nature of block ciphers, an error in the ciphertext only affects the plaintext belonging to the corresponding block, unless the blocks making up the message are chained in a way that creates dependencies between the blocks. Both the block cipher proposed by Pan et al. [48] and the modified version discussed in Section 5.8 cause highly localized error propagation when an error occurs in the bit sequence represented by the arrows on the facelets of the Rubik's cube. However, each ciphertext is accompanied by a randomly selected turn sequence r, and an error in this turn sequence could result in vastly different results.

Lastly, message expansion is problematic with the schemes discussed here due to the addition of the randomly chosen turn sequence r to each block (see Section 5.7). None of the other block ciphers listed in Table 5.2 add overhead to each block of a message.

5.9.2 Confusion and diffusion

The attacks described in the previous sections can be seen as edge cases of statistical analysis. Shannon identifies two properties of ciphers that, in general, make statistical analysis more difficult. These properties are *confusion* and *diffusion*, where *confusion* means that it should be difficult to relate statistics obtained from ciphertexts to the secret key and *diffusion* means that small changes in the plaintext should result in large changes in the ciphertext.

Remark 5.18. All of the commonly used block ciphers listed in Table 5.2 (IDEA [37], Triple DES [29], Twofish [57], Rijndael [11], and Camellia [3]) were designed with confusion and diffusion in mind.

The scheme proposed by Pan et al. [48] lacks diffusion entirely because each bit in the plaintext corresponds to exactly one bit in the ciphertext, and changing its value in the plaintext does not affect any other bits in the ciphertext. However, due to the use of a randomly chosen turn sequence r for each encryption, this effect cannot be easily observed (without fixing r). It is still possible to use the lack of confusion to break the security of the scheme as shown, for example, in Section 5.5 because the orientation of each arrow on each facelet depends on the original orientation of the arrow on the same facelet only and not on the orientation of the arrows on any other facelets.
As Section 5.6 demonstrated, it is possible to recover the Rubik's cube's state through a chosen-plaintext attack. Arguably, this indicates a low degree of confusion in the scheme. Again, it is only due to the use of the randomly chosen turn sequence r that the key cannot be recovered directly from the Rubik's cube's state, but, as explained in Section 5.6.2, the computational complexity of recovering the key from the Rubik's cube state does not necessarily warrant intractability assumptions.

5.10 Outlook

In this section, we will briefly consider the effect of the flaws that were discovered here on two proposed commitment schemes, as well as what could processes have led to the discovery of these flaws during the design of the system.

5.10.1 Commitment schemes based on the same ideas

Earlier this year, Pan et al. published another paper [49] proposing new cryptographic commitment schemes (see Section 4.2 for a brief introduction to commitment schemes). The first of the two new schemes, $\mathfrak{C}_1$, is based on the symmetric encryption scheme that has been discussed in this chapter, and whose security is claimed to be directly related to the conjugacy problem again. However, it is immediately obvious that $\mathfrak{C}_1$ does not address any of the weaknesses of the underlying encryption scheme. In fact, using the techniques that were developed in this chapter, it appears that the commitment scheme can be broken rather easily. The second proposed scheme, $\mathfrak{C}_2$, is not based on the encryption scheme $\mathfrak{S}_1$, and an in-depth analysis is out of scope for this thesis. Nonetheless, the scheme also encodes the plaintext message onto the surface of a Rubik's cube in the form of arrows and then applies a sequence of rotations before decoding the arrows to form the commitment. Given that some of the techniques in this chapter work regardless of the sequence of turns that is applied to the cube, it seems inevitable that the scheme $\mathfrak{C}_2$ can also be broken using the same or similar methods.

5.10.2 Lessons learned

What lessons can researchers learn from the fatal flaws in the proposed encryption and commitment schemes? Finding flaws in a design or implementation can be challenging, and proving that a system is flawless is difficult if not impossible. Nevertheless, it seems that standard cryptanalytic approaches could have led to the discovery of significant weaknesses in this case.

In Section 5.9, we used some of the earliest theoretical foundations of modern cryptography to evaluate the proposed schemes. Even if we had not found any weaknesses in the previous sections, using only knowledge from the 1940s, we would have found the complete lack of diffusion (see Section 5.9.2), which is unusual for a block cipher, as well as message expansion (see Section 5.9.1), which is also untypical when compared to modern block ciphers. By themselves, these properties are not necessarily problematic, however, they should have warranted investigation.

The unusual design, especially the requirement of a random rotating sequence r for each block, makes it difficult to evaluate some aspects of the algorithm. For example, symmetric encryption algorithms are often analyzed in the context of pseudorandom function families, however, due to the non-determinism of the proposed block cipher, it is difficult to apply such models. Virtually all common symmetric block ciphers are deterministic and only the *block cipher mode of operation*, such as cipher block chaining (CBC), counter (CTR), etc., adds non-determinism to the system [18]. Adopting such a design allows the separation of non-determinism from the security of the cipher itself during analysis.

However, despite these difficulties in the analysis of the proposed schemes, some standard cryptanalytical methods would have inevitably led to the discovery of weaknesses. For example, it is common to observe the propagation of bits from the plaintext to the internal state of the algorithm and possibly even to the resulting ciphertext. The designers of the Rijndael cipher, which ended up becoming the Advanced Encryption Standard (AES), mention this technique as the basis of many cryptanalysis methods [11]. Their design ensures that, after only two of the at least ten internal rounds within their cipher, every bit of the internal cipher state depends on every input bit, thus making it extremely difficult to trace the propagation of a single bit.

Applying the same cryptanalytic method to the cipher proposed by Pan et al. would have, at the very least, revealed not only the lack of diffusion but also the propagation of a part of the plaintext regardless of the secret key, allowing partial plaintext recovery attacks as described in Section 5.3. Similarly, the same analysis method would have shown that all input bits can only affect a subset of the output bits as depicted, for example, in Figure 5.7.

Provable security, that is, the reduction of a presumably hard problem to an attack against a cryptographic system, is an area of great importance. Admittedly, the proof published by Pan et al. [48] is presented convincingly. However, in this case, a mistake in the reduction from the conjugacy problem for the Rubik's cube group to an IND-CPA attack against the encryption scheme invalidates the security proof entirely. Even if we had not found practical attacks against the scheme, this would still mean a lack of provable security.

Finally, Pan et al. focus on the performance of their proposed scheme and compare it to that of established encryption schemes, such as Rijndael [11] in CBC mode (AES-CBC) and OFB mode (AES-OFB) and even asymmetric algorithms, without addressing the fundamental differences between those and their own schemes [48]. Additionally, Pan et al. do not account for hardware differences between benchmarks from as early as 2009 and their own measurements. Performance is an important factor when designing cryptographic algorithms, however, the performance evaluation presented by Pan et al. [48] is a misleading comparison at best, and fails to account for basic differences between algorithms and benchmarks.

Chapter 6

Other related ideas

This chapter will briefly list a few noteworthy ideas for other cryptographic mechanisms that are based on (or related to) Rubik's cubes, a discussion of which would be out of scope for this thesis. For an in-depth discussion of a zero-knowledge protocol [70] based on Rubik's cubes, refer to Chapter 4. For the cryptanalysis of a broken symmetric encryption scheme based on Rubik's cubes [48] and its relation to presumably broken commitment schemes [49], refer to Chapter 5 and Section 5.10.1, respectively. The following selection does not represent endorsement.

6.1 Without reductions to group theoretical problems

The following proposals do not attempt to prove the security of the proposed schemes based on intractability assumptions derived from group theoretical problems.

6.1.1 Image encryption

Loukhaoukha et al. proposed two image encryption algorithms based on Rubik's cubes [15, 40] in 2012 and 2013. Internally, similar to the allowed turns of a Rubik's cube, rows and columns of the image's pixels are rotated. These domain-specific algorithms aim to be sufficiently fast for real-time image encryption and decryption at the cost of providing a smaller degree of security than commonly used general-purpose block ciphers.

6.1.2 Pseudo-random bit generation

Raza and Satpute suggested a pseudo-random bit generation algorithm that scrambles its internal state by arranging it on the surface of a Rubik's cube and applying a sequence of turns [51]. They relate the security of the system to *chaos theory*. In particular, small changes in the system's internal state are expected to cause large differences in the generated bit sequences.
The design involves the use of a cryptographic hash function for seeding the generator. Of course, a cryptographic hash function itself is sufficient for constructing a secure random bit generator [5], however, Raza and Satpute claim that their design is less computationally expensive [51, p. 6162].

6.2 With reductions to group theoretical problems

The following proposals are based on problems that have reductions to group theoretical problems, thus basing the security of the schemes on intractability assumptions.

6.2.1 Cryptographic hash functions

In their article *Rubik's for Cryptographers*, which was published in 2013, Christophe Petit and Jean-Jacques Quisquater discuss Cayley hash functions, which are based on problems within Cayley graphs of groups, and they describe these problems as generalizations of the Rubik's cube puzzle [50]. They conclude that, while such hash functions have been broken in the past due to weak parameters, there is no reason to disregard the existence of secure parameters as long as the underlying computational problems are presumed to be hard. The group representing the 3x3x3 Rubik's cube is certainly small enough to solve even optimally, however, generalizations to larger cubes and other noncommutative groups could be reasonable choices.

6.2.2 Key agreement and public-key cryptosystem

Multiple key exchange protocols have been proposed that are based on variants of the conjugacy problem (see Section 3.1). The most notable of these protocols is likely the Ko-Lee-Cheon-Han-Kang-Park (KLCHKP) key agreement system [32], which is sometimes called *Braid Diffie-Hellman* since it is based on the same ideas as the well-known Diffie-Hellman key exchange protocol [16] from 1976, but uses elements of braid groups instead of integers. Unlike the original Diffie-Hellman protocol, the braid group version was suspected to resist quantum computing attacks. Additionally, Ko et al. proposed a public-key cryptosystem based on the ideas behind their key agreement system.

However, in 2003, Cheon and Jun found a polynomial-time attack against the Ko-Lee-Cheon-Han-Kang-Park key agreement system [9]. While their algorithm runs in polynomial time, its complexity is still too large for practical attacks. Nevertheless, the existence of a polynomial-time method renders the scheme insecure from a theoretical point of view. The attack does not solve the general cases of the underlying group theoretical problems.

Multiple other key exchange protocols have been proposed based on similar ideas [1, 2, 67] as well as multiple authentication schemes [64, 65], and, while many of these have also been broken [44, 45, 63], it remains an interesting area of research [13, 21]. Additionally, the application of group theory to cryptography has led to new research avenues within group theory [46].

Chapter 7

Summary

Ernő Rubik might not have foreseen the enormous impact of his creation almost 50 years ago. Despite the simplicity of the design of the Rubik's cube and of its mathematical representation as a permutation group, we have seen that the challenges it poses are not limited to merely solving it by hand.

In Chapter 2, we have discussed a mathematical model that accurately represents Rubik's cubes of arbitrarily large sizes. Based on this permutation-based model, we presented selected computational problems that are directly related to Rubik's cubes and that can be generalized to other permutation groups in Chapter 3. For most of those problems, no polynomial-time algorithms are known, but their potential hardness has not been proven either. Surely, group theory encompasses a wide range of computational problems, and will likely remain an active research area for the foreseeable future.

In Chapter 4, we have discussed a zero-knowledge protocol whose design is based on Rubik's cubes, even though the mathematical models representing Rubik's cubes themselves do not result in ideal parameters. For example, while the permutation group representing the 5x5x5 Rubik's cube (the "Professor's cube") is complex enough to provide a sufficient level of security for the zero-knowledge protocol (see Section 4.6), the size of the group is small compared to the size of each permutation, leading to a high overhead. Nevertheless, Rubik's cubes allow an intuitive understanding of the protocol before we progress toward more efficient parameter sets in Section 4.7 and Section 4.8, which we constructed in a way that maximizes efficiency within fixed symmetric groups. In addition to this contribution, we also made an implementation of the protocol available online (see Section 4.9).

Next, in Chapter 5, we offered the cryptanalysis of a symmetric encryption scheme whose security was supposedly based on the conjugacy problem (see Section 3.1). However, we managed to not only find flaws in the theory of the design, but also developed practical attacks ranging from disproving IND-CPA security and partial plaintext recovery to ciphertext forgery and even secret key recovery. The weaknesses that were found diminish the security of the proposed scheme almost entirely, and we extended this result to two commitment schemes in Section 5.10.1.

Lastly, in Chapter 6, we briefly discussed other ideas for cryptosystems that are based on or related to Rubik's cubes. Of particular interest are those in Section 6.2, whose security assumptions are based on intractability assumptions of group theoretical problems. Unfortunately, many of those cryptosystems have been broken, which is why there is much disagreement as to whether group theory is a promising area for cryptographic research. Because the group axioms do not restrict the nature of groups much and because groups are such general objects within the field of abstract algebra, *group-based cryptography* has a wide variety of groups to choose from. Even more algebraic structures become available when extending the field to *noncommutative cryptography*. Most asymmetric cryptosystems that are in use today rely on number theory, which is based on commutative algebraic structures. Nevertheless, as we have seen in Chapter 4 and briefly discussed in Section 6.2, promising designs of noncommutative cryptosystems exist, even though many have been broken. On a positive note, every time we base the security of an algorithm on a presumably difficult mathematical problem, we learn something new about the problem, even more so if someone manages to break the cryptosystem.

Bibliography

[1] Iris Anshel, Michael Anshel, and Dorian Goldfeld. An algebraic method for public-key cryptography. *Mathematical Research Letters*, 6(3):287–291, 1999. DOI: https://doi.org/10.4310/MRL.1999.v6.n3.a3.

[2] Iris Anshel, Michael Anshel, Dorian Goldfeld, and Stephane Lemieux. Key agreement, the Algebraic Eraser$^{\mathrm{TM}}$, and lightweight cryptography. In Lothar Gerritzen, Dorian Goldfeld, Martin Kreuzer, Gerhard Rosenberger, and Vladimir Shpilrain, editors, *Algebraic Methods in Cryptography*, volume 418 of *Contemporary Mathematics*, pages 1–34. American Mathematical Society, 2006.

[3] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms — Design and Analysis. In Douglas R. Stinson and Stafford Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer, Berlin, Heidelberg, 2001. DOI: https://doi.org/10.1007/3-540-44983-3_4.

[4] Elaine Barker. Recommendation for Key Management: Part 1 - General. Technical Report NIST SP 800-57 Part 1 Rev. 5, National Institute of Standards and Technology, Gaithersburg, MD, May 2020. DOI: https://doi.org/10.6028/NIST.SP.800-57pt1r5.

[5] Elaine B. Barker and John M. Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Technical Report SP 800-90A Rev. 1, National Institute of Standards and Technology, June 2015. DOI: https://doi.org/10.6028/NIST.SP.800-90Ar1.

[6] Manuel Blum, Paul Feldman, and Silvio Micali. Non-Interactive Zero-Knowledge and Its Applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing - STOC '88*, pages 103–112. Association for Computing Machinery, 1988. DOI: https://doi.org/10.1145/62212.62222.

[7] Peter J. Cameron. *Permutation Groups*. Number 45 in London Mathematical Society Student Texts. Cambridge University Press, Cambridge, 1999. DOI: https://doi.org/10.1017/CBO9780511623677.

[8] Arthur Cayley. Desiderata and Suggestions: No. 2. The Theory of Groups: Graphical Representation. *American Journal of Mathematics*, 1(2):174–176, 1878. DOI: https://doi.org/10.2307/2369306.

[9] Jung Hee Cheon and Byungheup Jun. A Polynomial Time Algorithm for the Braid Diffie-Hellman Conjugacy Problem. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, Lecture Notes in Computer Science, pages 212–225. Springer, Berlin, Heidelberg, 2003. DOI: https://doi.org/10.1007/978-3-540-45146-4_13.

[10] Stephen A. Cook. Can Computers Routinely Discover Mathematical Proofs? *Proceedings of the American Philosophical Society*, 128(1):40–43, 1984.

[11] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Information Security and Cryptography. Springer, Berlin, Heidelberg, 2002. DOI: https://doi.org/10.1007/978-3-662-04722-4.

[12] M. Dehn. Über unendliche diskontinuierliche Gruppen. *Mathematische Annalen*, 71(1):116–144, March 1911. DOI: https://doi.org/10.1007/BF01456932.

[13] Patrick Dehornoy. Braid-based cryptography. In Alexei G. Myasnikov and Vladimir Shpilrain, editors, *Group Theory, Statistics, and Cryptography*, volume 360 of *Contemporary Mathematics*, pages 5–33. American Mathematical Society, 2004.

[14] Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Anna Lubiw, and Andrew Winslow. Algorithms for Solving Rubik's Cubes. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms – ESA 2011*, volume 6942 of *Lecture Notes in Computer Science*, pages 689–700. Springer, Berlin, Heidelberg, 2011. DOI: https://doi.org/10.1007/978-3-642-23719-5_58.

[15] Adrian-Viorel Diaconu and Khaled Loukhaoukha. An Improved Secure Image Encryption Algorithm Based on Rubik's Cube Principle and Digital Chaotic Cipher. *Mathematical Problems in Engineering*, March 2013. DOI: https://doi.org/10.1155/2013/848392.

[16] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976. DOI: https://doi.org/10.1109/TIT.1976.1055638.

[17] John D. Dixon and Brian Mortimer. *Permutation groups*, volume 163 of *Graduate Texts in Mathematics*. Springer, New York, NY, 1996. DOI: https://doi.org/10.1007/978-1-4612-0731-3.

[18] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Technical Report NIST SP 800-38a, National Institute of Standards and Technology, Gaithersburg, MD, 2001. DOI: https://doi.org/10.6028/NIST.SP.800-38a.

[19] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, Berlin, Heidelberg, 1987. DOI: https://doi.org/10.1007/3-540-47721-7_12.

[20] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, Berlin, Heidelberg, 1999. DOI: https://doi.org/10.1007/3-540-48405-1_34.

[21] Lothar Gerritzen, Dorian Goldfeld, Martin Kreuzer, Gerhard Rosenberger, and Vladimir Shpilrain. *Algebraic Methods in Cryptography*, volume 418 of *Contemporary Mathematics*. American Mathematical Society, 2006. DOI: https://doi.org/10.1090/conm/418.

[22] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, July 1991. DOI: https://doi.org/10.1145/116825.116852.

[23] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC '85*, pages 291–304. Association for Computing Machinery, 1985. DOI: https://doi.org/10.1145/22145.22178.

[24] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. DOI: https://doi.org/10.1137/0218012.

[25] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219. Association for Computing Machinery, 1996. DOI: https://doi.org/10.1145/237814.237866.

[26] Shai Halevi and Silvio Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215. Springer, Berlin, Heidelberg, 1996. DOI: https://doi.org/10.1007/3-540-68697-5_16.

[27] Mike Hamburg, Paul Kocher, and Mark E. Marson. Analysis of Intel's Ivy Bridge Digital Random Number Generator. Technical report, Cryptography Research, Inc., San Francisco, CA, March 2012.

[28] David Joyner. *Adventures in Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys*. Johns Hopkins University Press, Baltimore, MD, 2nd edition, 2008.

[29] P. Karn, P. Metzger, and W. Simpson. The ESP Triple DES Transform. RFC 1851, RFC Editor, 1995. DOI: https://doi.org/10.17487/RFC1851.

[30] Graham Kendall, Andrew Parkes, and Kristian Spoerer. A Survey of NP-complete Puzzles. *ICGA Journal*, 31(1):13–34, 2008. DOI: https://doi.org/10.3233/ICG-2008-31103.

[31] Donald E. Knuth. *Seminumerical algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Boston, 3rd edition, 1997.

[32] Ki Hyoung Ko, Sang Jin Lee, Jung Hee Cheon, Jae Woo Han, Ju-sung Kang, and Choonsik Park. New Public-Key Cryptosystem Using Braid Groups. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 166–183. Springer, Berlin, Heidelberg, 2000. DOI: https://doi.org/10.1007/3-540-44598-6_10.

[33] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19, 2019. DOI: https://doi.org/10.1109/SP.2019.00002.

[34] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, Berlin, Heidelberg, 1996. DOI: https://doi.org/10.1007/3-540-68697-5_9.

[35] Herbert Kociemba. Cube Explorer 5.14, 2019. `http://kociemba.org/cube.htm`. Accessed on 2021-08-30.

[36] Richard E. Ladner. On the Structure of Polynomial Time Reducibility. *Journal of the ACM*, 22(1):155–171, 1975. DOI: https://doi.org/10.1145/321864.321877.

[37] Xuejia Lai and James L. Massey. A Proposal for a New Block Encryption Standard. In Ivan Bjerre Damgård, editor, *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, Berlin, Heidelberg, 1991.

[38] Edmund Landau. Über die Maximalordnung der Permutationen gegebenen Grades. *Archiv der Mathematik und Physik*, 3(5):92–103, 1903.

[39] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium* (*USENIX Security 18*), pages 973–990. USENIX Association, August 2018.

[40] Khaled Loukhaoukha, Jean-Yves Chouinard, and Abdellah Berdai. A Secure Image Encryption Algorithm Based on Rubik's Cube Principle. *Journal of Electrical and Computer Engineering*, March 2012. DOI: https://doi.org/10.1155/2012/173931.

[41] Alexander Lubotzky. *Discrete Groups, Expanding Graphs and Invariant Measures*, volume 125 of *Progress in Mathematics*. Birkhäuser, Basel, 1994. DOI: https://doi.org/10.1007/978-3-0346-0332-4.

[42] Charles F. Miller III. *On Group-Theoretic Decision Problems and Their Classification*, volume 68 of *Annals of Mathematics Studies*. Princeton University Press, 1971.

[43] Charles F. Miller III. Decision Problems for Groups — Survey and Reflections. In Gilbert Baumslag and Charles F. Miller III, editors, *Algorithms and Classification in Combinatorial Group Theory*, volume 23 of *Mathematical Sciences Research Institute Publications*, pages 1–59. Springer, New York, NY, 1992. DOI: https://doi.org/10.1007/978-1-4613-9730-4_1.

[44] Natalia Mosina and Alexander Ushakov. Mean-set attack: cryptanalysis of Sibert et al. authentication protocol. *Journal of Mathematical Cryptology*, 4(2):149–174, 2010. DOI: https://doi.org/10.1515/jmc.2010.006.

[45] Alex D. Myasnikov and Alexander Ushakov. Cryptanalysis of the Anshel-Anshel-Goldfeld-Lemieux Key Agreement Protocol. *Groups Complexity Cryptology*, 1(1):63–75, 2009. DOI: https://doi.org/10.1515/GCC.2009.63.

[46] Alexei Myasnikov, Vladimir Shpilrain, and Alexander Ushakov. *Non-commutative Cryptography and Complexity of Group-theoretic Problems*, volume 177 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2011. DOI: https://doi.org/10.1090/surv/177.

[47] National Institute of Standards and Technology. The Keyed-Hash Message Authentication Code (HMAC). Technical Report NIST FIPS 198-1, National Institute of Standards and Technology, Gaithersburg, MD, July 2008. DOI: https://doi.org/10.6028/NIST.FIPS.198-1.

[48] Ping Pan, Yun Pan, Zhen Wang, and Licheng Wang. Provably Secure Encryption Schemes With Zero Setup and Linear Speed by Using Rubik's Cubes. *IEEE Access*, 8:122251–122258, 2020. DOI: https://doi.org/10.1109/ACCESS.2020.3007335.

[49] Ping Pan, Junzhi Ye, Yun Pan, Lize Gu, and Licheng Wang. New Commitment Schemes Based on Conjugacy Problems over Rubik's Groups. *Information*, 12(8), 2021. DOI: https://doi.org/10.3390/info12080294.

[50] Christophe Petit and Jean-Jacques Quisquater. Rubik's for Cryptographers. *Notices of the American Mathematical Society*, 60(6):733–739, 2013. DOI: https://doi.org//10.1090/noti1001.

[51] Saiyma Fatima Raza and Vishal R. Satpute. PRaCto: Pseudo Random bit generator for Cryptographic application. *KSII Transactions on Internet and Information Systems*, 12(12):6161–6176, December 2018. DOI: https://doi.org/10.3837/tiis.2018.12.029.

[52] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978. DOI: https://doi.org/10.1145/359340.359342.

[53] Tomas Rokicki and Morley Davidson. God's Number is 26 in the Quarter-Turn Metric, 2014. `https://cube20.org/qtm/`. Accessed on 2021-08-31.

[54] Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge. God's Number is 20, 2010. `https://cube20.org/`. Accessed on 2021-08-31.

[55] Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge. The Diameter of the Rubik's Cube Group Is Twenty. *SIAM Journal on Discrete Mathematics*, 27(2):1082–1105, 2013. DOI: https://doi.org/10.1137/120867366.

[56] Ernő Rubik, Tamás Varga, Gerzson Kéri, György Marx, and Tamás Vekerdy. *Rubik's Cubic Compendium*. Number 3 in Recreations in Mathematics. Oxford University Press, New York, 1987.

[57] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-Bit Block Cipher. *First Advanced Encryption Standard (AES) Candidate Conference*, 1998.

[58] Uwe Schöning. Graph isomorphism is in the low hierarchy. In Franz J. Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors, *STACS 87*, volume 247, pages 114–124. Springer, Berlin, Heidelberg, 1987. DOI: https://doi.org/10.1007/BFb0039599.

[59] Ákos Seress. *Permutation Group Algorithms*. Number 152 in Cambridge Tracts in Mathematics. Cambridge University Press, 2003. DOI: https://doi.org/10.1017/CBO9780511546549.

[60] Claude E. Shannon. A Mathematical Theory of Cryptography. Bell System Technical Memo MM 45-110-02, September 1945.

[61] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4):656–715, October 1949. DOI: https://doi.org/10.1002/j.1538-7305.1949.tb00928.x.

[62] Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Computer Society, 1994. DOI: https://doi.org/10.1109/SFCS.1994.365700.

[63] Vladimir Shpilrain. Cryptanalysis of Stickel's Key Exchange Scheme. In Edward A. Hirsch, Alexander A. Razborov, Alexei Semenov, and Anatol Slissenko, editors, *Computer Science – Theory and Applications*, volume 5010 of *Lecture Notes in Computer Science*, pages 283–288. Springer, Berlin, Heidelberg, 2008. DOI: https://doi.org/10.1007/978-3-540-79709-8_29.

[64] Vladimir Shpilrain and Alexander Ushakov. An Authentication Scheme Based on the Twisted Conjugacy Problem. In Steven M. Bellovin, Rosario Gennaro, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, volume 5037 of *Lecture Notes in Computer Science*, pages 366–372. Springer, Berlin, Heidelberg, 2008. DOI: https://doi.org/10.1007/978-3-540-68914-0_22.

[65] Hervé Sibert, Patrick Dehornoy, and Marc Girault. Entity authentication schemes using braid word reduction. *Discrete Applied Mathematics*, 154(2):420–436, February 2006. DOI: https://doi.org/abs/10.5555/1167813.1705206.

[66] David Singmaster. *Notes on Rubik's Magic Cube*. Enslow Publishers, Hillside, New Jersey, 1981.

[67] E. Stickel. A New Method for Exchanging Secret Keys. In *Third International Conference on Information Technology and Applications (ICITA'05)*, volume 2, pages 426–430. IEEE, 2005. DOI: https://doi.org/10.1109/ICITA.2005.33.

[68] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.3)*, 2021. `https://www.sagemath.org`.

[69] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. DOI: https://doi.org/10.1112/plms/s2-42.1.230.

[70] Emmanuel Volte, Jacques Patarin, and Valérie Nachef. Zero Knowledge with Rubik's Cubes and Non-abelian Groups. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *Cryptology and Network Security*, volume 8257 of *Lecture Notes in Computer Science*, pages 74–91. Springer, Cham, 2013. DOI: https://doi.org/10.1007/978-3-319-02937-5_5.

[71] Daniel Walton. rubiks-cube-NxNxN-solver, April 2017. `https://github.com/dwalton76/rubiks-cube-NxNxN-solver`.

[72] World Cube Association. Records, 2021. `https://www.worldcubeassociation.org/results/records`. Accessed on 2021-08-30.

[73] World Cube Association. WCA Regulations, March 2021. `https://www.worldcubeassociation.org/regulations/`. Accessed on 2021-08-30.

Appendix A

Generators for the Professor's cube

For completeness, this section lists the generators of the permutation group representing the 5x5x5 Rubik's cube (see Chapter 2).

$$
\begin{aligned}
U_0 \coloneqq\ & (1\ 5\ 24\ 20)\ (2\ 10\ 23\ 15)\ (3\ 14\ 22\ 11)\ (4\ 19\ 21\ 6) \\
& (7\ 9\ 18\ 16)\ (8\ 13\ 17\ 12) \\
& (25\ 97\ 73\ 49)\ (26\ 98\ 74\ 50)\ (27\ 99\ 75\ 51)\ (28\ 100\ 76\ 52)\ (29\ 101\ 77\ 53) \\
U_1 \coloneqq\ & (30\ 102\ 78\ 54)\ (31\ 103\ 79\ 55)\ (32\ 104\ 80\ 56)\ (33\ 105\ 81\ 57)\ (34\ 106\ 82\ 58) \\
L_0 \coloneqq\ & (25\ 29\ 48\ 44)\ (26\ 34\ 47\ 39)\ (27\ 38\ 46\ 35)\ (28\ 43\ 45\ 30) \\
& (31\ 33\ 42\ 40)\ (32\ 37\ 41\ 36) \\
& (1\ 49\ 121\ 120)\ (6\ 54\ 126\ 115)\ (11\ 59\ 131\ 110)\ (15\ 63\ 135\ 106)\ (20\ 68\ 140\ 101) \\
L_1 \coloneqq\ & (2\ 50\ 122\ 119)\ (7\ 55\ 127\ 114)\ (12\ 60\ 132\ 109)\ (16\ 64\ 136\ 105)\ (21\ 69\ 141\ 100) \\
F_0 \coloneqq\ & (49\ 53\ 72\ 68)\ (50\ 58\ 71\ 63)\ (51\ 62\ 70\ 59)\ (52\ 67\ 69\ 54) \\
& (55\ 57\ 66\ 64)\ (56\ 61\ 65\ 60) \\
& (20\ 73\ 125\ 48)\ (21\ 78\ 124\ 43)\ (22\ 83\ 123\ 38)\ (23\ 87\ 122\ 34)\ (24\ 92\ 121\ 29) \\
F_1 \coloneqq\ & (15\ 74\ 130\ 47)\ (16\ 79\ 129\ 42)\ (17\ 84\ 128\ 37)\ (18\ 88\ 127\ 33)\ (19\ 93\ 126\ 28)
\end{aligned}
$$

$$
\begin{aligned}
R_0 :=\ & (73\ 77\ 96\ 92)\,(74\ 82\ 95\ 87)\,(75\ 86\ 94\ 83)\,(76\ 91\ 93\ 78) \\
& (79\ 81\ 90\ 88)\,(80\ 85\ 89\ 84) \\
& (24\ 97\ 144\ 72)\,(19\ 102\ 139\ 67)\,(14\ 107\ 134\ 62)\,(10\ 111\ 130\ 58)\,(5\ 116\ 125\ 53) \\
R_1 :=\ & (23\ 98\ 143\ 71)\,(18\ 103\ 138\ 66)\,(13\ 108\ 133\ 61)\,(9\ 112\ 129\ 57)\,(4\ 117\ 124\ 52) \\
B_0 :=\ & (97\ 101\ 120\ 116)\,(98\ 106\ 119\ 111)\,(99\ 110\ 118\ 107)\,(100\ 115\ 117\ 102) \\
& (103\ 105\ 114\ 112)\,(104\ 109\ 113\ 108) \\
& (5\ 25\ 140\ 96)\,(4\ 30\ 141\ 91)\,(3\ 35\ 142\ 86)\,(2\ 39\ 143\ 82)\,(1\ 44\ 144\ 77) \\
B_1 :=\ & (10\ 26\ 135\ 95)\,(9\ 31\ 136\ 90)\,(8\ 36\ 137\ 85)\,(7\ 40\ 138\ 81)\,(6\ 45\ 139\ 76) \\
D_0 :=\ & (121\ 125\ 144\ 140)\,(122\ 130\ 143\ 135)\,(123\ 134\ 142\ 131)\,(124\ 139\ 141\ 126) \\
& (127\ 129\ 138\ 136)\,(128\ 133\ 137\ 132) \\
& (44\ 68\ 92\ 116)\,(45\ 69\ 93\ 117)\,(46\ 70\ 94\ 118)\,(47\ 71\ 95\ 119)\,(48\ 72\ 96\ 120) \\
D_1 :=\ & (39\ 63\ 87\ 111)\,(40\ 64\ 88\ 112)\,(41\ 65\ 89\ 113)\,(42\ 66\ 90\ 114)\,(43\ 67\ 91\ 115)
\end{aligned}
$$

Appendix B

Reference implementation of the zero-knowledge protocol in Python

We present an implementation of the generic zero-knowledge protocol proposed by Volte et al. [70], which has been discussed in Chapter 4. The implementation consists of three Python classes. It makes the same assumptions as the protocol definition itself.
Additionally, to ensure compatibility with SageMath [68], which is based on Python, this implementation requires that `a * b` evaluates to $b \circ a$ when a and b are both permutations (see Remark 2.2).
The only required import is the `randbelow` function from the `secrets` library, which is used to generate the prover's secret key $i_1, \ldots, i_d$ and to choose the verifier's question q.

```
from secrets import randbelow
```

The `SecretKey` class is used to encapsulate a secret key $i_1, \ldots, i_d$. An instance of the class is held by the prover, who can use the function `get_public_key` to obtain the public key, which identifies the prover. The `get_generator_conjugate` function is provided for convenience only and not meant to be used directly.

```
class SecretKey:
    def __init__(self, params, secret_key):
        self.params = params
        self.secret_key = secret_key

    def get_public_key(self):
        p = self.params
        pub = p.F[self.secret_key[0]]
        for i in self.secret_key[1:]:
            pub = pub * p.F[i]
        return pub ** -1

    def get_generator_conjugate(self, j, tau):
        p = self.params
        i = self.secret_key[j - 1]
        return (tau ** -1) * p.F[i] * tau
```

The `Proof` class represents a single round from the perspective of the prover. It is instantiated from within the `Protocol` class. The prover can use the `commitments` function to obtain the commitments that must be sent to the verifier. Finally, the prover uses the `answer` function to obtain the answer for a question `q` that was provided by the verifier.

```
class Proof:
    def __init__(self, secret_key, tau, sigma, k_star, k):
        self.secret_key = secret_key
        self.tau = tau
        self.sigma = sigma
        self.k_star = k_star
        self.k = k
        self.answered = False

    def commitments(self):
        p = self.secret_key.params
        c0 = p.commit(self.k_star, self.tau)
        s = [p.commit(self.k[i], self.sigma[i]) for i in range(p.d + 1)]
        return (c0, s)

    def answer(self, q):
        if self.answered:
            raise Exception('answer may only be called once')
        self.answered = True
        if q == 0:
            d = self.secret_key.params.d
            return (self.tau, self.sigma[0], self.k_star, self.k[0], self.k[d])
        else:
            f_tau = self.secret_key.get_generator_conjugate(q, self.tau)
            return (f_tau, self.sigma[q], self.k[q - 1], self.k[q])
```

Lastly, the `Protocol` class provides the high-level functions for both the prover and the verifier. The prover invokes the `generate_key` function to generate a new secret key, from which the public key (and thus identity) can be derived (see above for the definition of `get_public_key` in the `SecretKey` class). The prover can obtain an instance of the `Proof` class by calling `begin_proof`. The verifier can use `choose_q` to randomly select a question, and then `check_answer` to verify that the received answer is valid.

```python
class Protocol:
    def __init__(self, params):
        self.params = params

    def generate_key(self):
        p = self.params
        return SecretKey(p, [randbelow(p.alpha) for i in range(p.d)])

    def begin_proof(self, secret_key):
        p = self.params
        tau = p.H.random_element()
        k_star = p.K.random_element()
        k = [p.K.random_element() for i in range(p.d + 1)]
        sigma = [p.G_.random_element()] + [None] * p.d
        for j in range(1, p.d + 1):
            f_tau = secret_key.get_generator_conjugate(j, tau)
            sigma[j] = (f_tau ** -1) * sigma[j - 1]
        return Proof(secret_key, tau, sigma, k_star, k)

    def choose_q(self):
        return randbelow(self.params.d + 1)

    def check_answer(self, commitments, q, answer, public_key):
        p = self.params
        (c0, s) = commitments
        if q == 0:
            (tau, sigma_0, k_star, k_0, k_d) = answer
            sigma_d = (tau ** -1) * public_key * tau * sigma_0
            assert tau in p.H
            assert p.commit(k_star, tau) == c0
            assert p.commit(k_0, sigma_0) == s[0]
            assert p.commit(k_d, sigma_d) == s[p.d]
        else:
            (f_tau, sigma_q, k_q_minus_1, k_q) = answer
            sigma_q_minus_1 = f_tau * sigma_q
            assert f_tau in p.F
            assert s[q - 1] == p.commit(k_q_minus_1, sigma_q_minus_1)
            assert s[q] == p.commit(k_q, sigma_q)
```

Using these classes requires a compatible definition of the system parameters `params`. Once such parameters are defined, usage follows this pattern:

```
zkp = Protocol(params)

# Invoked by the prover once to create a new identity.
secret_key = zkp.generate_key()
public_key = secret_key.get_public_key()

for round in range(r):
    # Prover:
    proof = zkp.begin_proof(secret_key)
    commitments = proof.commitments()
    # Verifier:
    q = zkp.choose_q()
    # Prover:
    answer = proof.answer(q)
    # Verifier:
    zkp.check_answer(commitments, q, answer, public_key)
```

This implementation is designed as a reference only and, after defining the necessary parameters, can help verify the correctness of other implementations. We have made an implementation written in the C programming language available online at

`https://github.com/tniessen/zkp-volte-patarin-nachef-c`.