Zero Knowledge with Rubik's Cubes

No Author Given

No Institute Given

Abstract. Since the invention of the Rubik's cube by Ernö Rubik in 1974, similar puzzles have been produced, with various number of faces or stickers. We can use these toys to define several problems in computer science, such as go from one state of the puzzle to another one. In this paper, we will classify some of these problems based on the classic Rubik's cube or on generalized Rubik's Cube. And we will see how we can use them in Zero Knowledge Authentication with a public key in order to achieve a given complexity against the best known attacks (for example 2^{80} computations). The efficiency of these schemes, and their possible connection with NP-complete problems will also be discussed.

Key words: Zero-Knowledge, Rubik's cube, authentication, symmetric group, cryptographic protocol, NP-completude

1 Introduction

The puzzles based on the Rubik's cube meet a great success. Generally speaking, Rubik's cube's owners try to solve the following problem: how to rearrange the cube when all the colors have been mixed up. At first sight, this problem seems very difficult, but there exist efficient algorithms to solve it. Nevertheless, several other problems with the cube and its neighboring puzzles seem to be really difficult from a computing point of view, as we will see later in this article. For example, if we impose that the number of moves is equal to a fixed value d that makes unique or almost unique the moves that must be done to recover the cube.
Consequently, we can try to build some public key Zero Knowledge authentication or signature protocols with a proven security linked to these difficult problems (and also on a hash function used for the commitments). It is well known that there exist cryptographic algorithms transforming every NP problem into a Zero Knowledge authentication protocol (cf [3]). The theorical way to do this is polynomial but nevertheless generally not efficient at all. This is why we will present and study some specific algorithms in this article, which can be used for practical cryptography and with a proven security based on some difficult well-known problems of the Rubik's cube.

2 Mathematical background

All the following notations can be found in the famous book [6], which is a funny way to learn algebra with toys such as the Rubik's cube and the Merlin's machine.

For a finite set X, S_X is the symmetric group of X. In the particular case $X = \{1; 2; \ldots; n\}$ where $n \in \mathbb{N}^*$, we call this group S_n. For $(\sigma, \sigma') \in {S_X}^2$, we use the classic notation $\sigma\sigma'$ to design the composition $\sigma' \circ \sigma$.
When G is a group, and $(g_1, g_2, \ldots, g_\alpha) \in G^\alpha$, $< g_1, g_2, \ldots, g_\alpha >$ is the subgroup generated by $g_1, g_2, \ldots g_\alpha$

Definition 1 *If G is a group and X a set, we say G acts on X (on the right) if there exists $\phi: G \to S_X$ such that:*
1. $\phi(e) = Id_X$
2. $\forall(\sigma, \sigma') \in G^2, \quad \phi(\sigma\sigma') = \phi(\sigma') \circ \phi(\sigma)$

For $x \in X$ and $\sigma \in G$, we can write $\phi(\sigma)(x) = \sigma * x$. Moreover, the orbit of x under G is $G * x = \{g * x | g \in G\}$

Example 1: When $G \subset S_n$, G acts on S_n in the following natural way:

$$\forall\sigma \in G, \forall\sigma' \in S_n, \quad \sigma * \sigma' = \sigma'\sigma$$

We will call this action **natural** right action.

3 Difficult Problems with Fairy Rubik

For all the following problems, we have a finite permutation group G, a finite set X, that represents the different states of the puzzle, and a right action of G on X. Let $\mathcal{F}$ be a subset of G, containing all authorized permutations.
The cardinal of $\mathcal{F}$ is $\alpha \geq 2$, we have $\mathcal{F} = \{f_1; f_2; \ldots; f_\alpha\}$.
$id \in X$ is the initial state.

Problem 1: *Resolving the puzzle.*
Given $x_0 \in X$, find $d \in \mathbb{N}$ and $(i_1, i_2, \ldots, i_d) \in \{1; 2; \ldots; \alpha\}^d$ such that

$$f_{i_1} f_{i_2} \ldots f_{i_d} * x_0 = id$$

Problem 2: *Go from one state to another one.*
Given $(x_0, x_1) \in X^2$, find $d \in \mathbb{N}$ and $(i_1, i_2, \ldots, i_d) \in \{1; 2; \ldots; \alpha\}^d$ such that

$$f_{i_1} f_{i_2} \ldots f_{i_d} * x_0 = x_1$$

Problem 3: *Resolving the puzzle in a given number of moves.*
Given $d \in \mathbb{N}^*$, $x_0 \in X$, find $(i_1, i_2, \ldots, i_d) \in \{1; 2; \ldots; \alpha\}^d$ such that

$$f_{i_1} f_{i_2} \ldots f_{i_d} * x_0 = id$$

Problem 4: *Go from one state to another with a given number of moves.*
Given $d \in \mathbb{N}^*$, $(x_0, x_d) \in X^2$, find $(i_1, i_2, \ldots, i_d) \in \{1; 2; \ldots; \alpha\}^d$ such that

$$f_{i_1} f_{i_2} \ldots f_{i_d} * x_0 = x_d$$

Proposition 1 *For the natural right action, problems 1 and 2 are equivalent, and so are problems 3 and 4.*

Proof. It is obvious that problems 1 and 3 are particular cases of problems 2 and 4 respectively.
Now, we suppose given $(x_0, x_1) \in X^2$ and we want to go from x_0 to x_1 with functions in $\mathcal{F}$. Let $x'_0 = x_1^{-1} x_0$. If we find $(i_1, \ldots, i_d)$ such that:

$$x'_0 f_{i_1} f_{i_2} \ldots f_{i_d} = f_{i_1} f_{i_2} \ldots f_{i_d} * x'_0 = id$$

then

$$x_0 f_{i_1} f_{i_2} \ldots f_{i_d} = f_{i_1} f_{i_2} \ldots f_{i_d} * x_0 = x_1$$

Proposition 2 *We can find a solution of problem 4 with $2\alpha^{d/2}$ computations if d is even.*

Proof. This is an attack in the middle. We notice that $f_{i_1} f_{i_2} \ldots f_{i_d} * x_0 = x_d$ is equivalent to $f_{i_1} \ldots f_{i_{d/2}} * x_0 = (f_{i_d})^{-1} \ldots (f_{i_{d/2+1}})^{-1} * x_d$.
So, for each $(i_1, i_2, \ldots, i_{d/2}) \in \{1, \ldots, \alpha\}^{d/2}$ we compute

$$\begin{aligned} & Y_{i_1 i_2 \ldots i_{d/2}} = f_{i_1} f_{i_2} \ldots f_{i_{d/2}} * x_0 \\ \text{and} \quad & Z_{i_1 i_2 \ldots i_{d/2}} = (f_{i_1})^{-1} (f_{i_2})^{-1} \ldots (f_{i_{d/2}})^{-1} * x_d \end{aligned}$$

Then we look for a collision between Y and Z.

Remark We give here (α, d) when the complexity of the attack is equal to 2^{80}

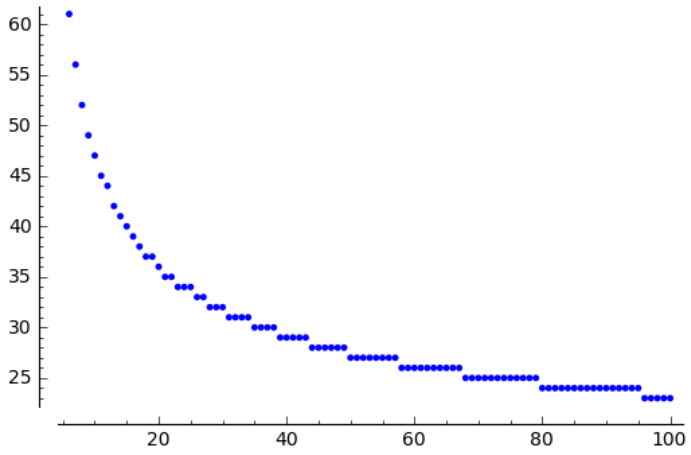
α	2	3	4	5	6	8	10	12
d	158	99	79	68	61	52	47	44

See fig. 1 for greater values of α.
In this paper we will study how to transform these difficult problems into a Zero-Knowledge Authentication Protocol. In other words: we will study how to prove that we have a solution of one of these problems without revealing anything of the solution.
Remark: The possible connection between NP complete, NP Space and these Rubik problems will be discussed in Appendix A.

4 Other notations

When X is a finite set, $\boxed{x \in_R X}$ means that we take a random element in X with an uniform probability.
A **commitment** is a function used to give somebody an hidden information, with the possibility of revealing it later. It's like putting something into a box and lock it, with the possibility of giving the key later. We use the notation $Com(x)$ to design a commitment of x. One easy way to construct such functions is the use of a hash function (see [5]).

Fig. 1. (α, d) for $\alpha \geq 6$ and a complexity equal to 2^{80} computations.



Definition 2 *For $\sigma \in S_N$, the **support** of σ is $supp(\sigma) = \{i \,|\, \sigma(i) \neq i\}$.*

Conjugation: Let G be a group, the conjugation on G defined by

$$\forall(\sigma, \tau) \in G^2, \quad \phi(\tau)(\sigma) = \tau^{-1}\sigma\tau = \sigma^\tau$$

is an action of G onto itself. Moreover we have:

$$\forall(\sigma, \sigma', \tau, \tau') \in G^4, \quad (\sigma^\tau)^{\tau'} = \sigma^{\tau\tau'}, \quad \sigma^\tau \sigma'^\tau = (\sigma\sigma')^\tau$$

We can also write $\boxed{\sigma^G = \{\sigma^g | g \in G\}}$.

5 With Rubik's cube $3 \times 3 \times 3$

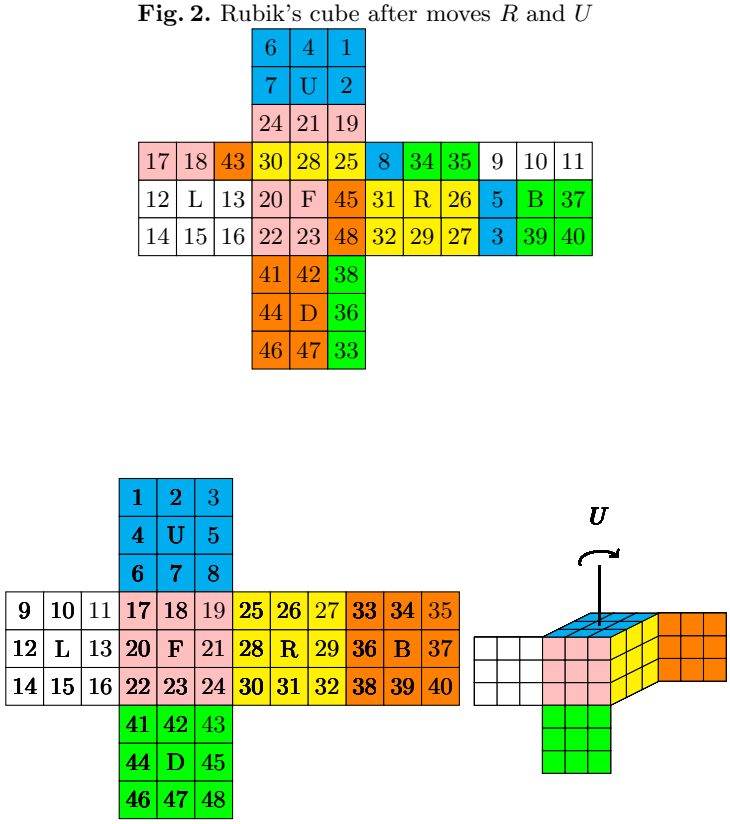
5.1 Introduction

We will first describe a zero-knowledge authentication scheme based on Rubick's classical cube $3 \times 3 \times 3$. We do this in order to introduce the main ideas with this relatively simple example. However with Rubik's cube $3 \times 3 \times 3$ the complexity of problem 4 is much smaller than 2^{80} and therefore we cannot use it for cryptographic security (for cryptographic applications we will use the Cube $5 \times 5 \times 5$ as we will see below).
We have indeed about 43.2×10^{18} different positions for this Rubik's cube, so about 2^{61} or 6^{25}. Tomas Rokicki (see http://tomas.rokicki.com/) pretends that we need less than 29 quarter turns to solve any position of the cube. If we consider that half a turn count as one move, we know that the God's number is 20 (see http://cube20.org/). But in our case we do not authorize un-clockwise quarter turns and half turns. So it seems reasonable to choose for problem 4 the value $d = 24$, and the security will be about $6^{12} \approx 2^{30}$ computations.

5.2 Mathematical representation of the Rubik's cube

For the Rubik's cube, we can write a number on each facets except the centers:

Fig. 2. Rubik's cube after moves R and U



then we define 6 permutations of S_{48} which are the basic clockwise quarter turns of the faces:

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

after the move σ, the facet $i \in \{1, 2, \ldots, 48\}$ is at the $\sigma(i)$ position.
The Rubik's cube group is $< F, B, L, R, U, D > \subset S_{48}$. If you want simulate the Rubik's cube, an easy way to do this is using SAGE [9].
For the Rubik's Cube, we will choose the natural right action, with $G = X = S_{48}$ and $\mathcal{F} = \{F, B, L, R, U, D\}$. So $\alpha = |\mathcal{F}| = 6$. If $x_0 \in X$ is the

initial position of the cube, you can visualize the cube by drawing the facet i at the position $x_0(i)$ for all $i \in \{1; \ldots; 48\}$. The Rubik's cube is solved when $x_d = id$.

5.3 Hiding the secret

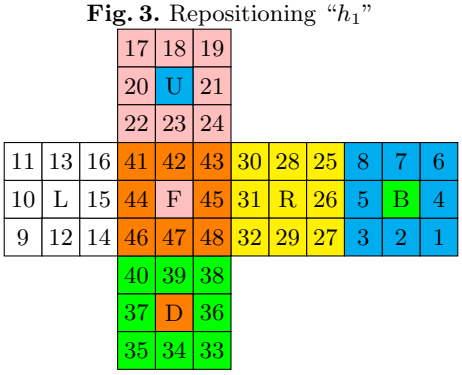
First we have to hide the permutation we make to go from state to another, without hiding that we make one authorized permutation, i.e. one element of $\mathcal{F}$. An easy way to do this with the cube $3 \times 3 \times 3$ is to roll the cube like a dice (we always consider that the centers of the faces don't move or don't exist). We will formalize this notion later, and call it the repositioning.
Let H be the group of the orientation-preserving symmetry of the cube. We have $H =< h_1, h_2 >$ where h_1 is the cube rolling on its back (see fig 3), and h_2 the cube laying on the table but turning as a whole one clockwise quarter of a turn. To be more precise we have:

$$h_1 = RL^{-1}(2, 39, 42, 18)(7, 34, 47, 23)$$
$$h_2 = UD^{-1}(13, 37, 29, 21)(12, 36, 28, 20)$$

It's easy to check that $|H| = 24$, because for each face up, we have 4 choices for the face in front.
Now let $x_0 \in X$ be one position of the cube. If $f \in_R \mathcal{F}$ and $\tau \in_R H$, we can check that f^τ is a random variable with an uniform law on $\mathcal{F}$, and the following diagram is commutative:

$$\begin{array}{ccc} x_0 & \xrightarrow{f} & x_1 \\ {\scriptstyle\tau}\downarrow & & {\scriptstyle\tau}\downarrow \\ \tau * x_0 & \xrightarrow{f^\tau} & \tau * x_1 \end{array}$$

Fig. 3. Repositioning "h_1"



Secondly, we want to hide any intermediate position of the cube, when we solve the problem. Physically, it is possible by removing all the stickers

of the cube (again, except the centers) and replacing them in a random way on the cube. We will actually do something close to that action at each step of the resolution.

5.4 ZK protocol

The prover's secret is $(i_1, i_2, \ldots, i_d) \in \{1, 2, \ldots, \alpha\}^d$ such that from the public $x_0 \in X$, we obtain $x_d \in X$ in exactly d rounds (d is also public) with the functions of the public set $\mathcal{F} = \{f_1, f_2, \ldots, f_\alpha\}$:

$$f_{i_1} f_{i_2} \ldots f_{i_d} * x_0 = x_d$$

We define for all $k \in \{1; \ldots; d-1\}$, $x_k = f_{i_k} * x_{k-1}$. The prover take $\tau \in_R H$ and $(\sigma_0, \sigma_1, \ldots, \sigma_d) \in_R {G'}^{d+1}$ where G' is the subgroup generated by $\mathcal{F}$ and H (we can imagine that the permutations σ_i are the removing-replacing stickers actions). Then he computes for all $k \in \{1; \ldots; d\}$ $g'_k = {\sigma_{k-1}}^{-1}({f_{i_k}}^{\tau})\sigma_k$. If we define for all $i \in \{0; \ldots; d\}$, $x'_i = \tau\sigma_i * x_i$, we have the following commutative diagram:

$$\begin{array}{ccccccccc}
x_0 & \xrightarrow{f_{i_1}} & x_1 & \xrightarrow{f_{i_2}} & \ldots & x_{d-1} & \xrightarrow{f_{i_d}} & x_d \\
\downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle \tau} & & & \downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle \tau} \\
\tau * x_0 & \xrightarrow{{f_{i_1}}^{\tau}} & \tau * x_1 & \xrightarrow{{f_{i_2}}^{\tau}} & \ldots & \tau * x_{d-1} & \xrightarrow{{f_{i_d}}^{\tau}} & \tau * x_d \\
\downarrow{\scriptstyle \sigma_0} & & \downarrow{\scriptstyle \sigma_1} & & & \downarrow{\scriptstyle \sigma_{d-1}} & & \downarrow{\scriptstyle \sigma_d} \\
x'_0 & \xrightarrow{g'_1} & x'_1 & \xrightarrow{g'_2} & \ldots & x'_{d-1} & \xrightarrow{g'_d} & x'_d
\end{array}$$

The prover send to the verifier $c_0 = Com(\tau)$, g'_k for all $k \in \{1; 2; \ldots; d\}$ and for all $i \in \{0; \ldots; d\}$, $s_i = Com(\sigma_i)$.
Then the verifier send to the prover $r \in \{0; 1; \ldots; d\}$.

If	prover reveals	Then the verifier checks
$r = 0$	$\tau,\ \sigma_0,\ \sigma_d$	$\tau\sigma_0 g'_1 g'_2 \ldots g'_d {\sigma_d}^{-1}\tau^{-1} * x_0 = x_d$ $\tau \in H$ Commitments c_0, s_0, s_d
$1 \leq r \leq d$	$\sigma_{r-1},\ \sigma_r$	$\sigma_{r-1} g'_r {\sigma_r}^{-1} \in \mathcal{F}$ Commitments s_{r-1}, s_r.

5.5 Proof of ZK protocol

Completeness Obviously, a legitimate prover will be always accepted.

Soundness If a prover can answer correctly in all cases, then he reveals $\tau \in H$, and σ_i for all $i \in \{0; 1; \ldots; d\}$.
So if we suppose that we have not found a collision in the hash function

used for commitment (see [5]), then we have:

$$\begin{aligned} x_d &= \tau\sigma_0 g'_1 \dots {g'_d}^{\sigma_d{}^{-1}}\tau^{-1} * x_0 \\ &= \tau(\sigma_0 g'_1 {\sigma_1}^{-1})(\sigma_1 g'_2 {\sigma_2}^{-1}) \dots ({\sigma_{d-1}} g'_d {\sigma_d}^{-1})\tau^{-1} * x_0 \\ &= \tau f_{i'_1} f_{i'_2} \dots f_{i'_d}\tau^{-1} * x_0 \\ &= {f_{i'_1}}^{\tau^{-1}} {f_{i'_2}}^{\tau^{-1}} \dots {f_{i'_d}}^{\tau^{-1}} * x_0 \end{aligned}$$

So we have a solution of the initial problem.

Black Box Zero-Knowledge Firstly we show that for a legitimate prover, each answer has an uniform probability over the corresponding set.

- $r = 0$.
  The prover gives $(\tau, \sigma_0, \sigma_d, g'_1, \dots g'_{d-1}) \in H \times {G'}^{d+1}$. We have for all $k \in \{1; \dots d-1\}$, $g'_k = {\sigma_{k-1}}^{-1} {f_{i_k}}^{\tau} \sigma_k$ with $\sigma_k \in_R G'$. So we have an uniform probability over $H \times {G'}^{d+1}$.
- $1 \leq r \leq d$.
  The prover gives $(\sigma_{r-1}, \sigma_r, {f_{i_r}}^{\tau}, g'_1, \dots, g'_{r-1}, g'_{r+1}, \dots, g'_d) \in {G'}^2 \times F \times {G'}^{d-1}$. We have for all $k \in \{1; \dots r-1\}$, $g'_k = {\sigma_{k-1}}^{-1} {f_{i_k}}^{\tau} \sigma_k$ with $\sigma_{k-1} \in_R G'$ and for all $k \in \{r+1; \dots d\}$, $g'_k = {\sigma_{k-1}}^{-1} {f_{i_k}}^{\tau} \sigma_k$ with $\sigma_k \in_R G'$. Moreover, since $\tau \in_R H$, ${f_{i_r}}^{\tau}$ is uniformly chosen in F (see below proposition 3). So we have again an uniform probability over ${G'}^2 \times F \times {G'}^{d-1}$.

Secondly, we construct a black-box simulator which take x_0 without knowing the secret, and interacts with a cheating verifier **CV**. We show that the simulator can impersonate the honest prover with probability $\frac{1}{d+1}$. The simulator randomly chooses a value $r^* \in_R \{0; 1; \dots; d\}$, this is a prediction what value **CV** will not choose. We consider two cases:

- $r^* = 0$
  We choose $\tau \in_R H$, $(f'_1, \dots, f'_d) \in_R F^d$ and $(\sigma_0, \dots, \sigma_d) \in_R {G'}^{d+1}$. Then we compute for all $k \in \{1; \dots; d\}$ $g'_k = {\sigma_{k-1}}^{-1} f'_k \sigma_k$.
- $1 \leq r^* \leq d$
  We choose $(i_1, i_2, \dots, i_{r^*-1}, i_{r^*+1}, \dots i_d) \in_R \{1; 2; \dots; \alpha\}^{d-1}$. Then we choose $\tau \in_R H$ and $(\sigma_0, \dots \sigma_d) \in_R {G'}^{d+1}$. We define for all $k \in \{1; \dots; r^*-1\}$, $x_k = f_{i_k} * x_{k-1}$, and for all $k \in \{r^*; \dots; d-1\}$, $x_k = {f_{i_{k+1}}}^{-1} * x_{k+1}$. We choose $f \in G$ such that $f * x_{r^*-1} = x_{r^*}$ (this is easy when $G = S_n$ and $F \subset S_n$). Then we compute for all $k \in \{1; \dots; d\} \backslash \{r^*\}$ $g'_k = {\sigma_{k-1}}^{-1} {f_{i_k}}^{\tau} \sigma_k$ and $g'_{r^*} = {\sigma_{k-1}}^{-1} g \sigma_k$ where $g = f^{\tau}$. We define also for all $i \in \{0; \dots; d\}$, $x'_i = \tau\sigma_i * x_i$. We have the following commutative diagram:

$$\begin{array}{ccccccccc}
x_0 & \xrightarrow{f'_1} & x_1 & \dots & x_{r^*-1} & \xrightarrow{f} & x_{r^*} & \dots\ x_{d-1} & \xrightarrow{f'_d} & x_d \\
\downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle \tau} & \downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle \tau} \\
\tau * x_0 & \xrightarrow{{f'_1}^{\tau}} & \tau * x_1 & & \tau * x_{r^*-1} & \xrightarrow{g} & \tau * x_{r^*} & \tau * x_{d-1} & \xrightarrow{{f'_d}^{\tau}} & \tau * x_d \\
\downarrow{\scriptstyle \sigma_0} & & \downarrow{\scriptstyle \sigma_1} & & \downarrow{\scriptstyle \sigma_{r^*-1}} & & \downarrow{\scriptstyle \sigma_{r^*}} & \downarrow{\scriptstyle \sigma_{d-1}} & & \downarrow{\scriptstyle \sigma_d} \\
x'_0 & \xrightarrow{g'_1} & x'_1 & \dots & x'_{r^*-1} & \xrightarrow{g'_{r^*}} & x'_{r^*} & \dots\ x'_{d-1} & \xrightarrow{g'_d} & x'_d
\end{array}$$

It is easy to check that each time every request of **CV** will have a satisfying answer, except for $r = r^*$. So the probability that it fails is $\frac{1}{d+1}$

5.6 Number of rounds

Here we will discuss of the number of times (r) the prover will do the protocol (one protocol is considered as one round), in order to prove with a good probability that she knows the secret. If we set this probability to $1 - 2^{-m}$, we must have $\left(\frac{d}{d+1}\right)^r \leq 2^{-m}$, so it gives $r \approx md\ln(2)$. For example, with $m = 30$ and $d = 24$, only 500 rounds are necessary, and it will take less than 1 second for a computer to compute everything.

6 Rubik's cube $5 \times 5 \times 5$

6.1 Introduction

Why choose this particular Rubik's cube ? For practical authentication we need a puzzle with at least 2^{160} different states. This cube has about 2^{247} different positions (see Wikipedia: Professor's cube). As for the Rubik's cube $4 \times 4 \times 4$, he has (only) about 2^{152} positions !

6.2 Mathematical representation

We can again write numbers on each facet, except the centers. But, on the contrary of the Rubik's cube $3\times3\times3$, two facets can be undistinguishable. So, when we observe this Rubik's Cube, we don't know exactly where the facets will go, and there may exist different ways to solve it. We can write this new problem in this way:

Problem 5: *Resolving the puzzle in a given number of moves with several initial states.*
Given $d \in \mathbb{N}^*$, $X_0 \subset X$, find $(i_1, i_2, \ldots, i_d) \in \{1; 2; \ldots; \alpha\}^d$, $x_0 \in X_0$ such that

$$f_{i_1} f_{i_2} \ldots f_{i_d} * x_0 = id$$

For the manipulation of the cube, we consider only 12 basic permutations. We will choose here the 6 clockwise quarter turns of the upper crown of each face (U, D, F, B, R, L), and the 6 clockwise quarter turns of the first intermediary crown of each face $(U_1, D_1, F_1, B_1, R_1, L_1)$. But other choices are possible. We have:

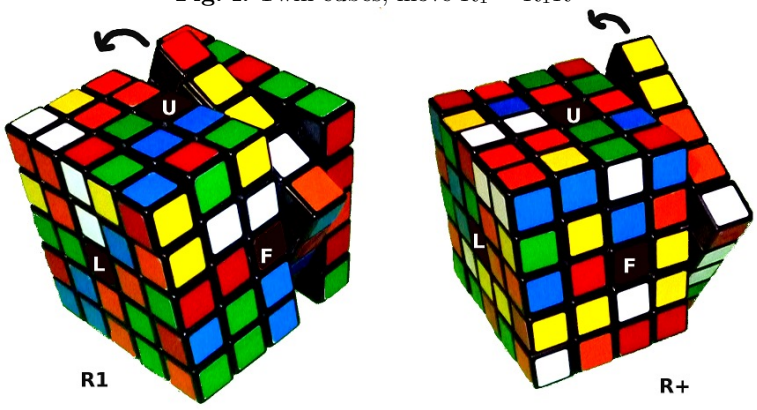
$$G = X =< U, D, \ldots, L, U_1, D_1, \ldots, L_1 > \subset S_{144}$$

6.3 Hiding the secret

Just rolling the cube is not enough to hide a authorized move. This will only shuffle independently (U, D, F, B, R, L) and $(U_1, D_1, F_1, B_1, R_1, L_1)$. We need a new idea. An elegant way to do this is to duplicate the cube. We have now 144 new facets, from 145 to 288. We will call U^+ the

action of U on the second cube, an easy way to compute U^+ is to add 144 to each cycle of U. We will call also E the exchange of the two cubes: $E = (1,145)(2,146)\dots(144,188)$. Then, each time we use U on the first cube, we will use U_1^+ on the second cube (i.e. $U_1 + 144$), and so on. Let's call $\tilde{G}$ the subgroup of S_{288} generated by UU_1^+, DD_1^+, ..., L_1L^+. The size of $\tilde{G}$ is about the same as G. A computation with Sage gives $|G| \approx 2^{300}$ and $\left|\tilde{G}\right| \approx 2^{364}$. We will hide the move by rolling in the same way the two cubes, and exchanging or no the two cubes. So we set $H =< h_1h_1^+, h_2h_2^+, E >$. This time our repositioning group H has 48 elements.

Fig. 4. Twin cubes, move $\tilde{R_1} = R_1R^+$



By the way, we have changed the problem to a new one:

Problem 6: *Resolving the puzzle in a given number of moves with several initial states and several final states.*
Given $d \in \mathbb{N}^*$, $(X_0, X_d) \subset X^2$, find $(i_1, i_2, \dots, i_d) \in \{1; 2; \dots; \alpha\}^d$, $x_0 \in X_0$, $x_d \in X_d$ such that

$$f_{i_1} f_{i_2} \dots f_{i_d} * x_0 = x_d$$

In this case X_0 the subset of all possible initial states with the given facets of the first cube (and all permutations for the second cube) and X_d the subset of all permutations with a support included in $\{145, \dots 288\}$. It is quite obvious that in this particular case, the second cube gives no information at all for the solution, so the problem has exactly the same difficulty than the initial problem. The only drawback is the size of the new objects, but we have seen that the size of $\tilde{G}$ is not so different as the size of G.

6.4 ZK Protocol

We can take the same scheme than in the previous section, with $\tilde{G}$ instead of G, and using problem 6. See annexe C for an overview of the protocol.

7 Generalisation with S_n

In this section, we will define precisely the repositioning group for a set of permutations $\mathcal{F}$. The existence of this group is the key of our schemes, as explained in the following proposition.

Definition 3 *Let $\mathcal{F} = \{f_1, \ldots, f_\alpha\} \subset S_n$. If there exists a subgroup $H \subset S_n$ such that ${f_1}^H = \mathcal{F}$ then H is called a **repositioning** group of $\mathcal{F}$.*

Proposition 3 *We suppose $\mathcal{F}$ has a repositioning group H. Let S_1 be the H-stabilizer of f_1 i.e. $S_1 = \{\tau \in H \mid {f_1}^\tau = f_1\}$, we have $|H| = |\mathcal{F}| \times |S_1|$. So, if we choose $\tau \in_R H$, $P({f_i}^\tau = f_j) = \frac{1}{\alpha}$ for all (i, j).*

As we have seen in the previous section, it's not easy to find a repositioning group in the general case. When the functions in $\mathcal{F}$ are not conjugated, it is even impossible. But we can use a roundabout way, with an extension set to find a solution. We will give here the general algorithm to build H. But we can notice that we can often find better solutions. For example, in the previous section, the orientation preserving group of the cube enables us to work on S_{288} instead of S_{1728}.
General method: Let $G = S_n$, $I_n = \{1; 2; \ldots n\}$, $\alpha \in \mathbb{N}$ $(\alpha \geq 2)$, $\mathcal{F} = \{f_1, \ldots, f_\alpha\}$. For all $k \in \{0, \ldots \alpha - 1\}$, we define $I_n^{(k)} = \{1 + kn; \ldots; (k+1)n\}$, and we consider that $f_k^{(i)}$ is the same permutation than f_k but on $I_n^{(i)}$ instead of I_n. We define for all $k \in \{1; 2; \ldots; \alpha\}$:

$$\tilde{f}_k = f_k^{(0)} f_{k+1}^{(1)} \ldots f_\alpha^{(\alpha-k)} f_1^{(\alpha-k+1)} \ldots f_{k-1}^{(\alpha-1)}$$

condidered as permutations of $I_{\alpha n}$. Let $\tilde{G} = S_{\alpha n}$ and $\sigma \in \tilde{G}$ defined by:

$$\forall i \in \{1; \ldots; \alpha n\}, \quad \sigma(i) = \begin{cases} i + n & \text{if } i \leq (\alpha - 1)n \\ i - (\alpha - 1)n & \text{if } i > (\alpha - 1)n \end{cases}$$

Then $H =< \sigma >$ is a repositioning group of $\tilde{\mathcal{F}} = \{\tilde{f}_1; \ldots; \tilde{f}_\alpha\}$.

8 Practical examples

In this section, we try to give a simple and efficient way to make a authentication protocol based on permutation. We do not start with the puzzle, but on the contrary, we construct our own puzzle directly with its repositioning group. The result is much more compact and can be used in practice.
We begin to set $G = S_{41}$ because $|G| \approx 2^{165}$. A concrete way to represent one element of this group is for example a playlist with only 41 different songs.
We take two random elements in G: f and h. We define $H =< h >$ and $\mathcal{F} = f^H$. With SAGE, we made a simulation with 1000 iterations (see

annexe B). 98 times over 100, we have $|< \mathcal{F} >| \geq \frac{|G|}{2}$. The mean value for $\alpha = |\mathcal{F}|$ is 468 with a standard deviance equal to 825. So there is no difficulty to find $\alpha \geq 100$.
When H, $\mathcal{F}$ and α are so defined, we set $d = 24$ (see fig 1), and $r = 500$. So we have a security equal to 2^{80} and a authentication scheme with an error less than 2^{-30}.

9 Conclusion

In this paper we have seen how we can build authentication schemes with public key, based on various problems linked on the Rubik's cube and several other generalized cubes. We also show how to construct a random puzzle over a small set that is suitable for the general scheme and can be used for a security in 2^{80}. It is also possible to transform these authentication schemes into signature schemes with the standard transformation used in the "Fiat-Shamir" protocol with a hash function (see [1]).
Our constructions are much more efficient than those obtained with general process (cf [3]). Other puzzles, not mentioned here, can be used in the same way for authentication, but there exist puzzles too much dissymmetric or based on some PSpace complete problems that would be worth having specific analysis.
We have also classified several problems linked to the generalized Rubik's cube and show that some NP complete problems have a neighboring description.

References

1. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – Eurocrypt 1990*, volume 473 of *Lecture Notes in Computer Science*, pages 481–486. Springer-Verlag, 1990.
2. Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completness.* W.H Freeman and Co, 2nd, 1991 edition, 1979.
3. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38:690–728, July 1991.
4. Oded Goldreich and Y. Oren. Definitions and properties if Zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.
5. Shai Halevi and Silvio Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing . In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215. Springer-Verlag, 1996.
6. David Joyner. *Adventures with Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys.* The Johns Hopkins University Press, 2nd edition, 2008.
7. Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A survey of np-complete puzzles. *ICGA Journal*, 31(1):13–34, 2008.

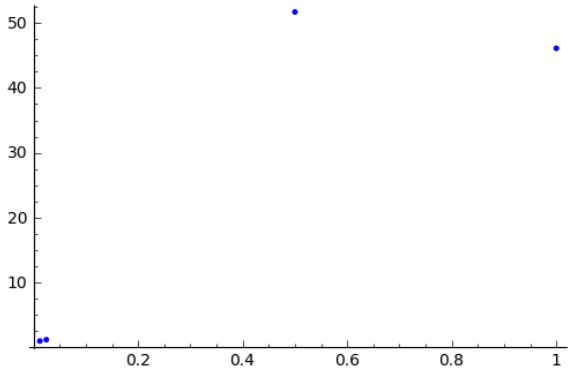
8. Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266, 1977.
9. W. A. Stein et al. *Sage Mathematics Software (Version 4.7-OSX-32bit-10.5)*. The Sage Development Team, 2011. `http://www.sagemath.org`.

A Possible connections between NP Complete, NP Space and Rubik Problems

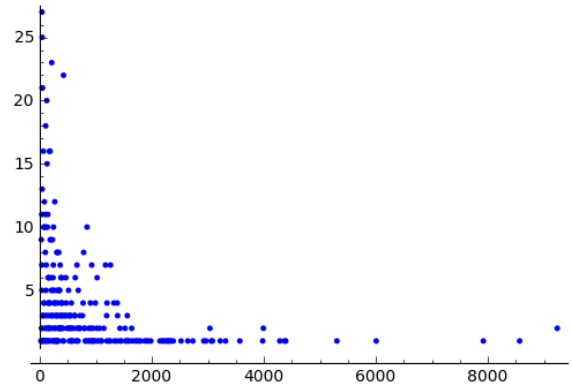
In section 3 we have seen several problems based on the Rubik's cube or on generalized Rubik's cubes. When one of the parameter of these puzzles (for example the size n of the cube) becomes large, we wonder how will grow the complexity, asymptotically speaking. We notice that we don't know yet if some of these problems are NP-complete (cf [7] p. 27). Moreover, it is plausible that they are not NP-complete because they have a power of description too much limited to describe all the problems of the NP class.
Nevertheless, as we will explain further, some of NP-complete problems have a real simililarity with the Rubik's cubes puzzles. So we can consider that these problems, used in this article for authentication, are part of a neighboring class, or a larger class, which is proved NP-complete or NP-space. This is not a proof of the difficulty of Rubik's cube related problems, but it is an indirect argument suggesting it could be true.
Example 1 From [2] p. 280 and [8] we know that the problem "Finite Function Generation" is P-space complete.
Finite Function Generation
INSTANCE: Finite set A, a collection F of functions $f: A \to A$ and a specified function $h: A \to A$.
QUESTION: Can h be generated from the functions in F by composition ?
Remark We can notice that here the number of composition functions to be found is not considered, unlike for the rubik's cube problems where this value d seems to be critical for the complexity.
Example 2 From [2] p. 213, we know that the problem "Longest path" is NP complete.
LONGEST PATH
INSTANCE: Graphe $G = (V, E)$, length $l(e) \in \mathbb{Z}^+$ for each $e \in E$, positive integer K, specified vertices $s, t \in V$
QUESTION: Is there a simple path in G from s to t of length K or more, i.e. whose edge lengths sum to at least K ?
Remark This problem remains NP complete if $l(e) = 1$ for all $e \in E$. Therefore this problem has some similarities with our Rubik problems for going from one position to another. However, as noticed in [2] p. 79 this problem becomes polynomial when we change "of length K or more" by "of length K or less". Nevertheless if me modelize our graph G such that each vertice is a position of a Rubik's cube $n \times n \times n$, the number of vertices (i.e. possible Cubes) will grow exponentially in n.

B Practical simulation

```
sage: set_random_seed(2)
sage: G=SymmetricGroup(41)
sage: gg=G.order()
sage: print "Group order:",gg,"->",int(ln(gg)/ln(2)+1),"bits"
sage: x=[]
sage: xx=[]
sage: nb_try=1000
sage: for j in range(nb_try):
...       f=G.random_element()
...       h=G.random_element()
...       n=h.order()
...       c=[h**i*f*h**(-i) for i in range(n)]
...       sc = set(c)
...       H=PermutationGroup(c)
...       hh=H.order()
...       x.append(hh)
...       if hh >= gg/2 :
...           xx.append(len(sc))
...
sage: sxx = Set(xx)
sage: sx = Set(x)
sage: print "y=P(card(<f_1,f-2,...,f_alpha>)/card S_n = x)"
sage: y = [(j/gg,100*x.count(j)/nb_try) for j in sx]
sage: show(points(y))
sage: print "alpha when <f_1,f_2...f_alpha>=S_n or A_n"
sage: y2 = [(j,xx.count(j)) for j in sxx]
sage: show(points(y2))
sage: print "mean value of alpha",float(mean(xx))
sage: print "standard deviance of alpha",\
float(sqrt(variance(xx)))
sage: print "mode of alpha",mode(xx)
Group order: 33452526613163807...2000000000 -> 165 bits
y=P(card(<f_1,f-2,...,f_alpha>)/card S_n = x)
```



alpha when <f_1,f_2...f_alpha>=S_n or A_n




```
mean value of alpha 468.853783231
standard deviance of alpha 825.082851267
mode of alpha [36]
sage: print "<F>=S_n :",\
100*float(x.count(gg)/nb_try),"%"
sage: print "<F>=A_n :",\
100*float(x.count(gg/2)/nb_try),"%"
<F>=S_n : 46.1 %
<F>=A_n : 51.7 %
```

C ZK general protocol

Public:

- A group G with a right action on a set X
- A subset $\mathcal{F} = \{f_1, \ldots, f_\alpha\} \subset G$
- A repositioning group $H \subset G$ such that $f_1^H = \mathcal{F}$.
- $d \in \mathbb{N}$, $d \geq 3$
- $(X_0, X_d) \subset X^2$.
- G' subgroup of G generated by $\mathcal{F}$ and H. $G' =< \mathcal{F}, H >$.

Secret:

$(i_1, i_2, \ldots, i_d) \in \{1, 2, \ldots, \alpha\}^d$ such that there exist $x_0 \in X_0$ and $x_d \in X_d$ with:

$$f_{i_1} f_{i_2} \ldots f_{i_d} * x_0 = x_d$$

Scheme (one round):

Prover		**Verifier**
$\tau \in_R H$ $\sigma_0, \sigma_1, \ldots, \sigma_d \in_R G'$ $\forall k \in \{1, \ldots, d\},\ g'_k = \sigma_{k-1}^{-1} f_{i_k}^{\tau} \sigma_k$ $c_0 = Com(\tau)$ $\forall i \in \{0, \ldots d\},\ s_i = Com(\sigma_i)$	$g'_1, \ldots, g'_d$ $c_0, s_0, \ldots, s_d$ $\longrightarrow$	
		$r \in_R \{0, \ldots d\}$
	r $\longleftarrow$	
If $r = 0$	$\tau,\ \sigma_0,\ \sigma_d$ $\longrightarrow$	**Checks** $\exists x_0, x_d \in X_0 \times X_d,$ $\tau\sigma_0 g'_1, g'_2 \ldots, g'_d \sigma_d^{-1} \tau^{-1} * x_0 \in X_d$ $\tau \in H$ $Com(\tau) = c_0,$ $Com(\sigma_0) = c_0,\ Com(\sigma_d) = c_d$ If all tests ok return "accepted" else return "rejected"
If $r \neq 0$	$\sigma_{r-1},\ \sigma_r$ $\longrightarrow$	**Checks** $\sigma_{r-1} g'_r {\sigma_r}^{-1} \in \mathcal{F}$ $s_{r-1} = Com(\sigma_{r-1}),\ s_r = Com(\sigma_r)$ If all tests ok return "accepted" else return "rejected"