Physical Zero-Knowledge Proof and NP-completeness Proof of Suguru Puzzle ⋆

Léo Robert, Daiki Miyahara, Pascal Lafourcade, Luc Libralesso, Takaaki Mizuki

Physical Zero-Knowledge Proof and NP-completeness Proof of Suguru Puzzle*

Léo Robert[a,*], Daiki Miyahara[b,d,*], Pascal Lafourcade[a,*], Luc Libralesso[a], Takaaki Mizuki[c,d,*]

[a] *University Clermont Auvergne, LIMOS, CNRS UMR 6158, Aubière France*
[b] *The University of Electro-Communications, Tokyo, Japan*
[c] *Cyberscience Center, Tohoku University, Sendai, Japan*
[d] *National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan*

Abstract

Suguru is a paper and pencil puzzle invented by Naoki Inaba. The goal of the game is to fill a grid with numbers between 1 and 5 while respecting three simple constraints. We first prove the NP-completeness of Suguru puzzle. For this we design gadgets to encode the PLANAR-CIRCUIT-SAT in a Suguru grid. We then design a physical Zero-Knowledge Proof (ZKP) protocol for Suguru. This ZKP protocol allows a prover to prove that he knows a solution of a Suguru grid to a verifier without leaking any information on the solution. To construct such a physical ZKP protocol, we only rely on a few physical cards and adapted encoding. For a Suguru grid with n cells, we only use $5n + 5$ cards. Moreover, we prove the three classical security properties of a ZKP: completeness, extractability, and zero-knowledge.

Keywords: Physical zero-knowledge proof, Suguru, Security, Completeness, Extractability, Zero-knowledge

1. Introduction

Zero-Knowledge Proofs (ZKP) were introduced in 1985 by Goldwasser et al. [10]. Two parties are involved in such a ZKP protocol: a prover P and a verifier V. At the end of the protocol, the verifier V is convinced that P knows the solution s to the instance $\mathcal{I}$ of a problem $\mathcal{P}$, without revealing any information about s. A zero-knowledge proof prevents the verifier from gaining any knowledge on the solution other than its correctness. When both randomization

*This article is an extended version of the paper published at SSS 2020 [29].

*Corresponding authors

Email addresses: leo.robert@uca.fr (Léo Robert), miyahara@uec.ac.jp (Daiki Miyahara), pascal.lafourcade@uca.fr (Pascal Lafourcade), luc.libralesso@uca.fr (Luc Libralesso), mizuki+tcs@tohoku.ac.jp (Takaaki Mizuki)

and interaction are allowed, the proofs that can be verified in polynomial time are exactly those that can be generated within polynomial space [34].

Formally, for a solution s to any instance $\mathcal{I}$ of a problem P, a convincing interactive zero-knowledge protocol between P and V must satisfy the following three properties[1]:

Completeness: If P knows s, then he is able to convince V.

Extractability[2]: If P does not know s, then he is not able to convince V except with a *small* probability. More precisely, we want a negligible probability, *i.e.*, the probability should be a function f of a security parameter λ (for example the number of repetitions of the protocol) such that f is negligible, that is for every polynomial Q, there exists $n_0 > 0$ such that:

$$\forall\ x > n_0, f(x) < \frac{1}{Q(x)}.$$

Zero-knowledge: V learns *nothing* about s except $\mathcal{I}$, *i.e.*, there exists a probabilistic polynomial time algorithm $\texttt{Sim}(\mathcal{I})$ (called the simulator) such that the outputs of the real protocol and those of $\texttt{Sim}(\mathcal{I})$ follow the same probability distribution.

There exist two kinds of ZKP: *interactive* and *non-interactive*. In an interactive ZKP, the prover can exchange messages with the verifier to convince him, while in the non-interactive case, the prover can only create the proof to convince the verifier.

ZKPs are usually executed by computers. They are often used in electronic voting to prove that some parties correctly mix some ballots without cheating, or in multi-party computation [5, 6, 30]. Moreover, there exist generic cryptographic zero-knowledge proofs for all problems in NP [9], via a reduction to an NP-complete problem with a known zero-knowledge proof.

In [28], the authors simply explained this concept to some children using a circular cave. This was the first proposition of a physical ZKP. Later, Gradwohl et al. [11] proposed a ZKP for the famous Nikoli's puzzle known as Sudoku[3]. They used some physical cards to construct a ZKP protocol. It was one of the first interactive physical ZKP protocols for such puzzles. We aim to design a ZKP protocol for Suguru puzzles in the same way as the one done for Sudoku.

Suguru was designed by Naoki Inaba[4], the original name of the game was *Nanba Burokku*, but it is also known as *Tectonics* or *Number Blocks*. Suguru is a paper and pencil puzzle in which a grid is divided into outlined blocks called

[1] Moreover, if $\mathcal{P}$ is NP-complete, then the ZKP should be run in polynomial time [8]. Otherwise, it might be easier to find a solution than proving that a solution is correct, making the proof pointless.

[2] This implies the standard soundness property, which ensures that if there exists no solution of the puzzle, then the prover is not able to convince the verifier with a small probability.

[3] `https://www.nikoli.co.jp/en/puzzles/sudoku.html`

[4] His website introduces many pencil puzzles (including Suguru) of his own making in Japanese: `http://www.inabapuzzle.com/`.

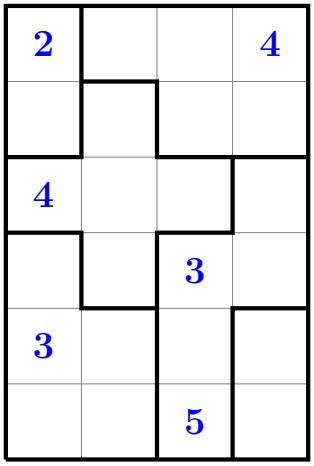


Figure 1: Initial Suguru grid

regions. Each region contains up to five cells. Every cell of the grid must contain a number from 1 to k (where $k \leq 5$ is the number of cells in the region). Each cell should be filled such that no two identical numbers coincide — not even diagonally.

Suguru's rule. This puzzle consists of a rectangular grid where blocks divide the overall area. These blocks called *regions* contain up to five cells. The goal is to fill all the cells with integers under the following constraints:

- **Number region rule:** A region composed of k cells must be filled with integers $1, \ldots, k$.
- **Neighbor rule:** For every cell, all of its eight neighbors must have different values from the cell's value.

In Figure 1, we give an example of an initial Suguru grid and in Figure 2, we give its unique solution.

Contributions. We study the computational complexity of Suguru and prove that Suguru is NP-complete. We propose a polynomial construction that allows us to encode the PLANAR-CIRCUIT-SAT problem that is a well-known NP-complete problem in the literature. For this we design logical gadgets for TRUE, FALSE, and also for the logical gates NOT, AND, and OR in rectangular Suguru grids of size 11×5. We also design compatible connectors to plug all these gadgets together. Hence we are able to encode any PLANAR-CIRCUIT-SAT problem in polynomial time in a Suguru grid, which prove that Suguru is NP-complete.

We also propose a simple ZKP protocol for Suguru using a small number of cards. Our construction is simple and can be used as a pedagogical example

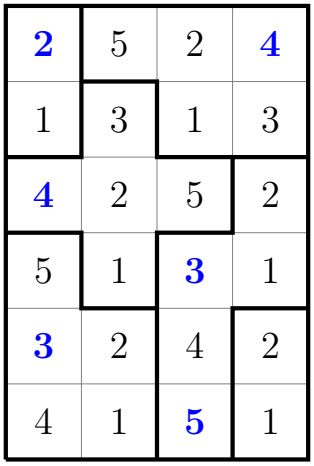


Figure 2: Solution of the Suguru grid of Figure 1

to explain the role of ZKP protocols. We propose an encoding of the number using simple cards. Using this encoding, the prover places some cards on the grid according to the solution. We use these cards to prove that the two rules of Suguru are satisfied. We start with the first number of a cell in a region; after verifying the validity of all regions and replacing the cards placed by the prover, we reuse them to prove the second rule of Suguru on the eight neighbors of each cell of the grid. Here is the difficulty of Suguru, because we need to prove that all values of the eight neighbors of each cell are different without revealing any information to the verifier. Here, we use a trick in our encoding of the values of the cards to avoid leaking any information. Our encoding requires five cards per cell; therefore, if a Suguru grid has n cells to guess, our protocol only requires $5n + 5$ cards. Finally, we prove the three security properties of our construction *i.e.,* completeness, extractability, and zero-knowledge.

Related work. In [33], the authors proposed an improved ZKP protocol for Sudoku that follows the pioneer work of [11]. In [7], the authors proposed a ZKP protocol for Nikoli's puzzle, Norinori.

In [15], a method to consider one feature of several puzzles that consists of constructing a single loop, was established. This technique used a topological approach with successive interactive transformations.

Recently several physical ZKP proofs have been proposed for different Nikoli's puzzles. In [21], card-based ZKP protocols for Takuzu and Juosan were proposed. In [31], a physical ZKP proof for Numberlink was designed. In [32], a ZKP protocol for Ripple Effect was designed. In [22], a card-based physical ZKP for Kakuro was established, improving the first version proposed by Bultel et al. in [3] with the ZKP protocols for three other Nikoli's games: Akari, Takuzu, and Kenken.

All these works clearly demonstrate that designing physical ZKP is an in-

teresting research topic. Each game has its own rules and requires an adapted construction.

Despite the existence of all those previous works, one cannot reuse or directly adapt them for the Suguru game. The main reason is the "strong" neighbor rule where no cell can have eight neighbors with the same value. Other puzzles have a similar rule but with relaxed restrictions. For instance, Makaro[5] has a neighbor rule but only for adjacent cells (and not in diagonal). Thus, a naive adaptation would imply a loss in terms of efficiency and zero-knowledge (no information about the solution can be leaked). Furthermore, it is worth noting that the encoding for the proof of NP-completeness of Makaro cannot be applied for Suguru. It is not easy to fulfill empty spaces between gadgets in Suguru because the number of possible values in the room is limited to 5 in Suguru while it is unlimited in Makaro. Moreover in Suguru, the neighbor rule with eight neigbors contraints more the construction for the empty parts of the grid and also connections between gadgets which make the task more tricky.

Many studies on secure computations with physical objects have been conducted. Using a deck of cards, we are able to solve Yao's millionaires' problem [20, 24, 25, 36] and efficiently compute any logical function [26, 27, 35]. Using PEZ dispensers, one can achieve secure multiparty computation [1, 2, 23].

This paper is an extended version of the SSS 2020 conference paper [29]; we mainly add the NP-completeness proof of Suguru, which solves an open question raised in the SSS 2020 paper.

Outline. In Section 2, we begin by a proof of NP-completeness of Suguru. In Section 3, we present our notations, and all sub-protocols needed to construct our ZKP. In Section 4, we design our ZKP protocol for Suguru. In Section 5, we prove the security of our protocol. In the last section, we conclude the paper.

2. Suguru is NP-complete

We present an NP-completeness proof for Suguru. The proof can be roughly summarized as follows: We reduce any PLANAR-CIRCUIT-SAT problem to solve a Suguru puzzle instance.

PLANAR-CIRCUIT-SAT is an extension of the satisfiability problem in which a formula can be represented by a circuit that is planar, meaning that the circuit can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its wires do not intersect and are only connected to the logical gates. It is a standard lecture notes question[6] to prove that this problem is NP-complete using the result that PLANAR-SAT is NP-complete [16]. Moreover, using PLANAR-CIRCUIT-SAT problem for proving NP-completeness is

[5] Makaro was proven to be NP-complete in [14].

[6] The reader can find the proof in Erik Demaine's lecture notes `http://courses.csail.mit.edu/6.892/spring19/scribe/lec6.pdf` or in Luca Trevisan's lecture notes `https://theory.stanford.edu/~trevisan/cs170/notes/lecture22.pdf`

often used in pen and pencil 2D puzzles. In the original version of PLANAR-CIRCUIT-SAT, the proof is done only with NAND operator, denoted by $\overline{\wedge}$. Since NAND is the inverse of an AND, it can be defined as follows: $\overline{(x \wedge y)}$. Notice that with only NAND it is possible to derivate all the other operators as follows:

- $\overline{x} = x \mathbin{\overline{\wedge}} x$
- $x \wedge y = (x \mathbin{\overline{\wedge}} y) \mathbin{\overline{\wedge}} (x \mathbin{\overline{\wedge}} y)$
- $x \vee y = (x \mathbin{\overline{\wedge}} x) \mathbin{\overline{\wedge}} (y \mathbin{\overline{\wedge}} y)$

In our proof, we have designed gadgets for OR, AND, and NOT which are more than enough to use the PLANAR-CIRCUIT-SAT NP-completeness result.

Running example. We use the following formula as our running example:

$$(\overline{x_1} \vee x_2) \wedge x_1.$$

This simple formula is composed of two clauses and contains two variables x_1 and x_2, OR, AND and NOT operators, respectively denoted in logics by $\vee$, $\wedge$, and $\overline{}$. Moreover, x_1 appears in both clauses. This formula can be represented by the plannar circuit given in Figure 3 where standard representation of OR, AND, and NOT gates are used.

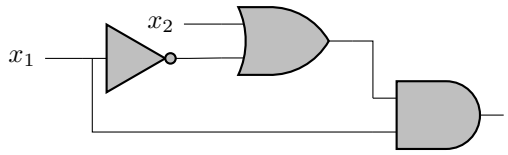


Figure 3: Circuit of our running example $(\overline{x_1} \vee x_2) \wedge x_1$.

The idea of our proof works as follows. We construct gadgets in Suguru, i.e., we first define a set of TRUE, FALSE (Section 2.1), then AND, OR, NOT, SPLIT gadgets (Section 2.2), the connectors/isolators to perform the wiring (Section 2.3), and finally the procedure to build the full Suguru instance that encodes any PLANAR-CIRCUIT-SAT formula (Section 2.4).

2.1. Modeling TRUE and FALSE

We first need to define a representation of TRUE and FALSE in a Suguru grid. We use the representation given in Figure 4 for TRUE and in Figure 5 for FALSE.

The shape of TRUE and FALSE is minimalist and uses only a room of size two. It is crucial, and it determines the design of all the other gadgets. It is important to notice that the semantics depends on the direction of reading of the circuit. It is why we add external arrows to help the reader to understand our gadgets in the paper. Of course these arrows are not part of our modeling. As we can see in Figures 4 and 5, TRUE encounters first 1 then 2 while FALSE meets first 2 and then 1.

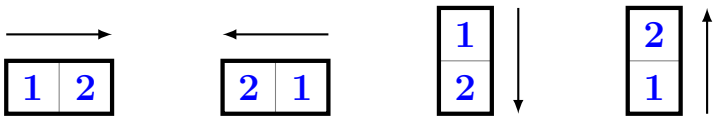


Figure 4: TRUE gadgets.

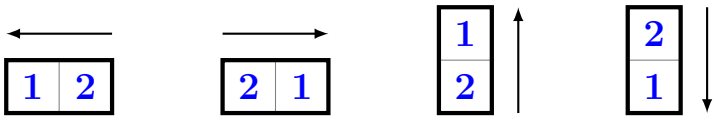


Figure 5: FALSE gadgets.

2.2. Gadgets for logical gates: NOT, AND, OR, and SPLIT

We give logical gadgets for the following logical gates: NOT, AND, OR, and SPLIT. All these gadgets are small Suguru grids of 5 lines and 11 columns. In order to help the reader to understand our gadgets, we add arrows to indicate the sense of TRUE and FALSE gadgets for the inputs. We do not add the arrow for the output, since it is clear enough with the coloring of the cells, due to the shape of our gadgets.

We start with the NOT that is the only one to have only one input and one output.

NOT gadget. In Figure 6, we present our NOT gadget.

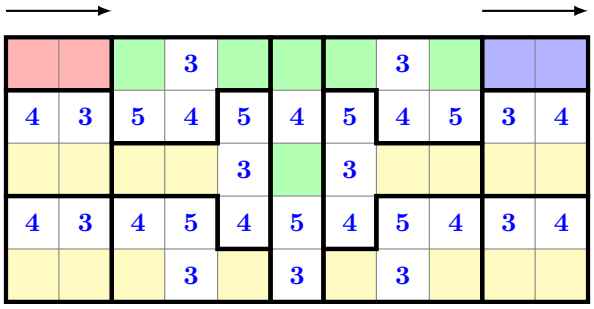


Figure 6: NOT gadget.

The input is represented by red cells, the output is represented by blue cells. Red and blue cells only take 1 or 2 depending on the current value (TRUE or FALSE) of the adjacent gadget of the input. In order to help the reader to understand our gadget, we paint in green cells that should contain 1 or 2 (called 1-2 cells in the rest of the paper) in our gadgets and that depend on the input. We also paint in yellow the 1-2 cells that are independent of the red and green 1-2 cells.

It is easy to observe that if TRUE is the value of the adjacent gadget of the red cell then 1 is the value in the leftmost red input cell, then the output

value of the NOT operator is 1 in the rightmost blue cell, which implies that the output is the inverse of the input; thus, the operator produces a FALSE. Similarly, if FALSE is the value of the adjacent gadget then 2 is the value of the leftmost input cell of the NOT operator, then its output in blue is the inverse of the input, which implies that the output is TRUE.

By inverting blue and red cells, we obtain a NOT gadget in the opposite direction (right to left).

OR gadget. In Figure 7, we present the OR gadget that takes two input roomss in red and outputs the result in the blue room.

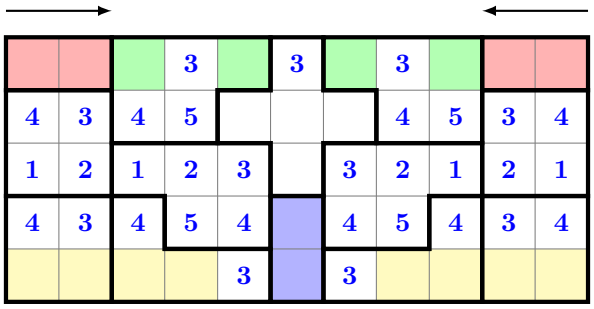


Figure 7: OR gadget

The positions of 1, 2, 3, 4, and 5 in the two white rooms of size 5 are crucial that are adjacent to the white room in the middle that forms a cross. We now prove that our OR gadget serves a logical OR.

- If the adjacent gadget of each of the two inputs is FALSE, i.e., the two red rooms contain 2 and 1 (in this order), then two green cells adjacent to the cross room contain 1. This implies that the bottom of the cross contains 1, and that the output blue two cells become 2 and 1, which means a FALSE value.
- If one of the adjacent gadgets of the two inputs is TRUE, i.e., one of the red rooms starts with 1, then at least one of the two green cells adjacent to the cross room contains 2. This implies that the 2 in the cross is in the bottom cell. This forces the output cells in blue to contain 1 and 2 (in this order) which means a TRUE value.

AND gadget. In Figure 8, we present the AND gadget. The only difference from the OR is that we swap 1 and 2 in the two white cells of size 5 adjacent to the cross, implying the need to swap to the 1 and 2 in the two adjacent cells of size 4. We emphasize on the difference by writing in red the values of the modified cells.

Our AND gadget works similarly to the OR gadget; here, the positions of the 1 close to the cross are crucial. We now prove that our AND gadget serves a logical AND.

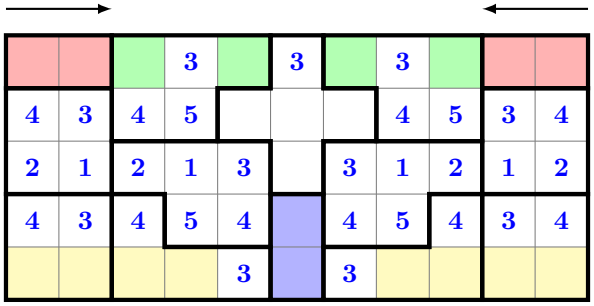


Figure 8: AND gadget

- In the case where the two inputs are TRUE, the two green cells adjacent to the cross on the first line contain 2. This implies that the 2 is in the bottom cell of the cross, the output cells in blue read 1 and 2, which is the TRUE value.
- If at least one of the inputs of the AND gadget is a FALSE, i.e., a red room starts with 2, then the 1 is in the bottom cell of the cross room; thus, the bottom-most output cell in blue contains 1, which is the FALSE value.

SPLIT Gadget. The last gadget required to design is a SPLIT gadget. This gadget is 22×5 cells as presented in Figure 9.

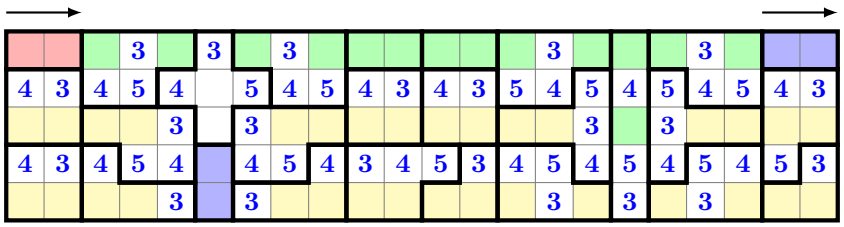


Figure 9: SPLIT gadget.

Our SPLIT gadget allows us to connect different elements. One can observe that it propagates its input (red room) to its outputs (blue rooms) of the same value (TRUE or FALSE). Notice that all the blue cells contain 1 or 2. We also emphasize that it can be use in any direction, since it just splits and propagates similar values on all directions.

To propagate the circuit above, we need to use a vertical propagator as we will define in Section 2.3.

An important remark is that NOT, OR, AND, and SPLIT gadgets have almost the same borders:

- the top and bottom lines contain only 1, 2 or 3 values;

- the leftmost and rightmost columns alternate 1 or 2 values and 4 value.

This structure in our gadgets helps us to design our connectors.

We also need to definie a SPLIT STOP gadet given in Figure 10 that allows us to propagate the circuit only to the top or bottom but stop the propagation on the right side. We put in red the 3 that has been moved to one cell to the right.

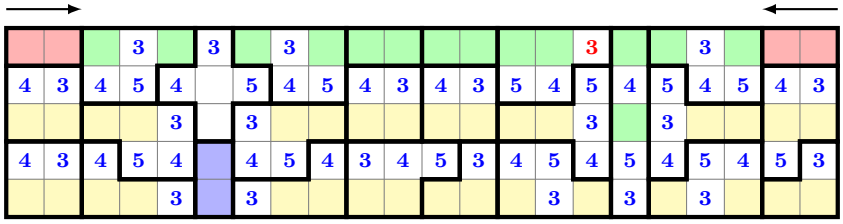


Figure 10: SPLIT STOP gadget.

2.3. Connecting and isolating "logic" components

We have introduced our "logical" gadgets. However, they cannot be directly connected. Thus, we introduce in this subsection horizontal and vertical connectors and isolators in order to plug all logical gadgets to construct a PLANAR-CIRCUIT-SAT formula.

More precisely, our idea is to construct a Suguru grid where there is an alternation between "logical" and "connection/isolation" layers, both vertically and horizontally.

We first present the horizontal isolator and the horizontal connector, before introducing the vertical isolator and the vertical connector.

Horizontal isolator. The horizontal isolator is described in Figure 11.

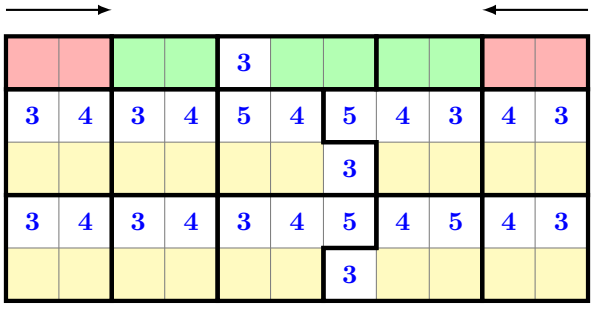


Figure 11: Horizontal isolator

The inputs on the left or right side are isolated thanks to the 3 in the middle of the first line and the line of 3, 4 and 5 below. The goal of this isolator gadget is not to propagate the 1 and 2 values.

Horizontal connector. Figure 12 presents our horizontal connector. In this figure, we present the case where the input is on the left side and the output is on the right side. To obtain a connector that works in the opposite direction, it is enough to swap the red and blue cells.

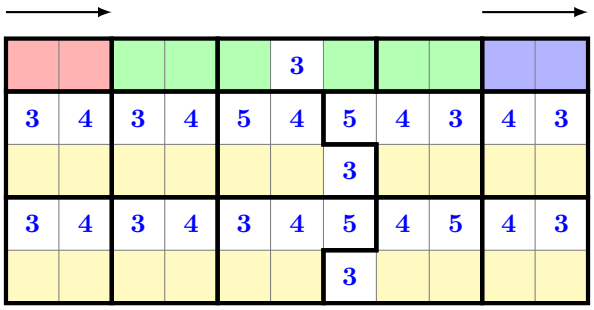


Figure 12: Horizontal connector

This connector allows values to propagate on the first row (in green), but not on the third and fifth rows (in yellow). Moreover, as all the logical gadgets have a 4 in their second and fourth rows, they are compatible with the horizontal connector (which has a 3 in its second and fourth rows). Besides, the horizontal yellow lines of this connector are disconnected, due to the vertical sequence of 5, 3, 5, 3; hence, they are compatible with any values 1 and 2. The horizontal connector has the same dimension as the logical gadgets (11×5).

Vertical isolator. The vertical isolator is presented in Figure 13.

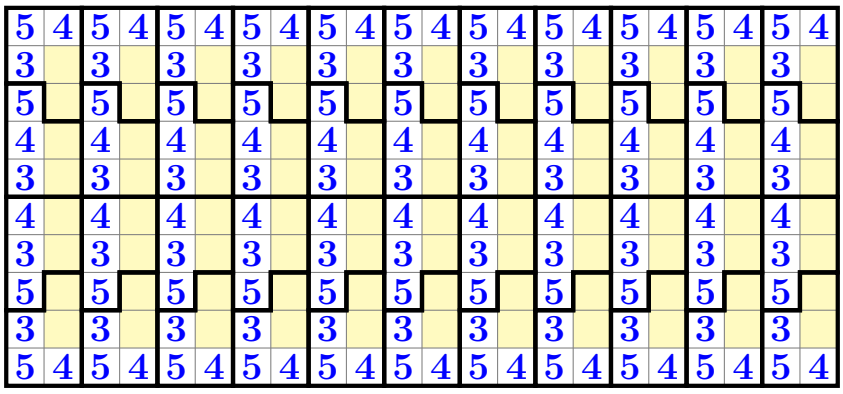


Figure 13: Vertical isolator

This behaves as the vertical connector except on the sixth column that does not expose a vertical logical gadget. As for the vertical connector, it is compatible horizontally with the vertical connectors/isolators, and is vertically compatible with the logical gadgets and horizontal connectors.

Vertical connector. The vertical connector is presented in Figure 14.

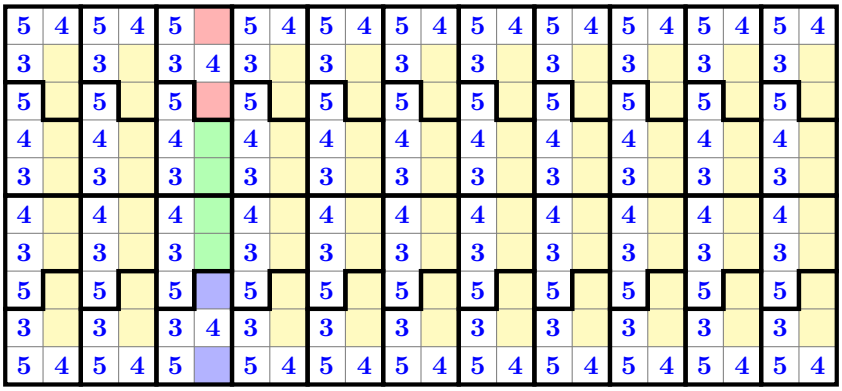


Figure 14: Vertical connector

It is vertically surrounded by 4s and 5s, making the first line and the last line compatible with our logical gadgets and horizontal connectors/isolators except at one position (sixth column). This trick allows us to connect the 1 and 2 between vertical layers. Notice that its size is (22×10), which is twice as large as the logical gadgets or horizontal connectors/isolators. Moreover, the vertical connector can be repeated as its left side has 5s in the first and 10th row, and 3s, 4s and 5s in the other rows while its right side has 4s in the first and 10th row and 1s and 2s in the other rows. By abuse of notations, we use the same colors (red, green and blue) to indicate clearly how this vertical connector works.

2.4. Building the complete Suguru instance

Having defined the logical gates (Section 2.2) and the connectors/isolators to build the wiring (Section 2.3), we are now able to build the complete Suguru instance.

We define a coarser grain grid, in which we alternate between a "logic gadget" and a "connector/isolator gadget". In other words, the first row of gadgets is composed of an alternation of "logic gadgets" and "horizontal connector/isolator gadgets". The second row of gadgets is composed of "vertical connector/isolators". The third row is composed of an alternation of "logic gadgets" and "horizontal connector/isolator gadgets", and so on. This allows to make sure that every gadget is compatible with its neighbors.

We fill each eventual gap using the SPLIT gadgets surrounded by the vertical isolators.

Let X be the maximum number of logical gadgets in a row of the planar circuit layout and Y be the maximum number of logical gadgets in a column of the planar circuit. The generated Suguru grid would have $X \times (11 + 11)$ rows and $Y \times (5 + 10) - 10$ columns, which are polynomial in the size of the input.

We need to use the SPLIT gadget if one variable appears serveral times in a formula, because we need to propagate its values into all the operators that are

using it. Since we use PLANAR-CIRCUIT-SAT, we are sure that we can always represent any formula into a Suguru grid with our gadgets. The transformation is clearly polynomial in the size of the circuit.

Encoding the running example. Using a planar embedding of our running example $((\overline{x_1} \vee x_2) \wedge x_1)$ given in Figure 3, we can position the logical gadgets and the connectors as follows by following the lines of the circuit:

x_1 SPLIT $\rightarrow$ $\downarrow$	$\rightarrow$ NOT $\rightarrow$	$\rightarrow$ OR x_2 $\downarrow$
SPLIT $\rightarrow$	$\rightarrow$ AND $\leftarrow$ $\downarrow$	$\leftarrow$ SPLIT

In a similar way to the existing NP-completeness proofs for other puzzles, e.g., [12, 17, 18, 19], when the cirucuit line is split, we use a SPLIT gadget and use the corresponding logical gadget and connector to obtain the corresponding Suguru grid.

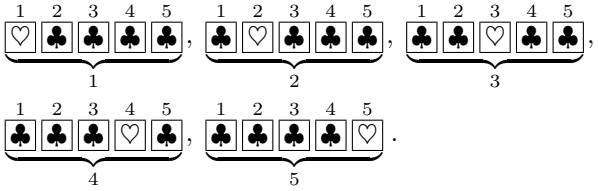
The result is given in Figure 15.

3. Preliminaries for the ZKP

From now on, we construct a physical ZKP protocol for a Suguru puzzle. For this, in this section, we introduce some card and shuffle notations used in our construction.

3.1. Notations

Card. A deck of cards used in our protocol consists of blacks $\boxed{\clubsuit}$ and reds $\boxed{\heartsuit}$ whose backs are identical $\boxed{?}$. Each integer $i \in \{1, \ldots, 5\}$ is encoded as:



We call such face-down five cards $\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}$ corresponding to an integer according to the above encoding rule a *commitment* to the respective integer. We also use *numbered* cards such as $\boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5}$ whose backs are identical $\boxed{?}$.

Neighbor cell. Consider a target cell denoted c_t on a grid. A cell is a *neighbor* of c_t if it is next to c_t. It can be on the left, right, top, or bottom of c_t, and also on its diagonal:

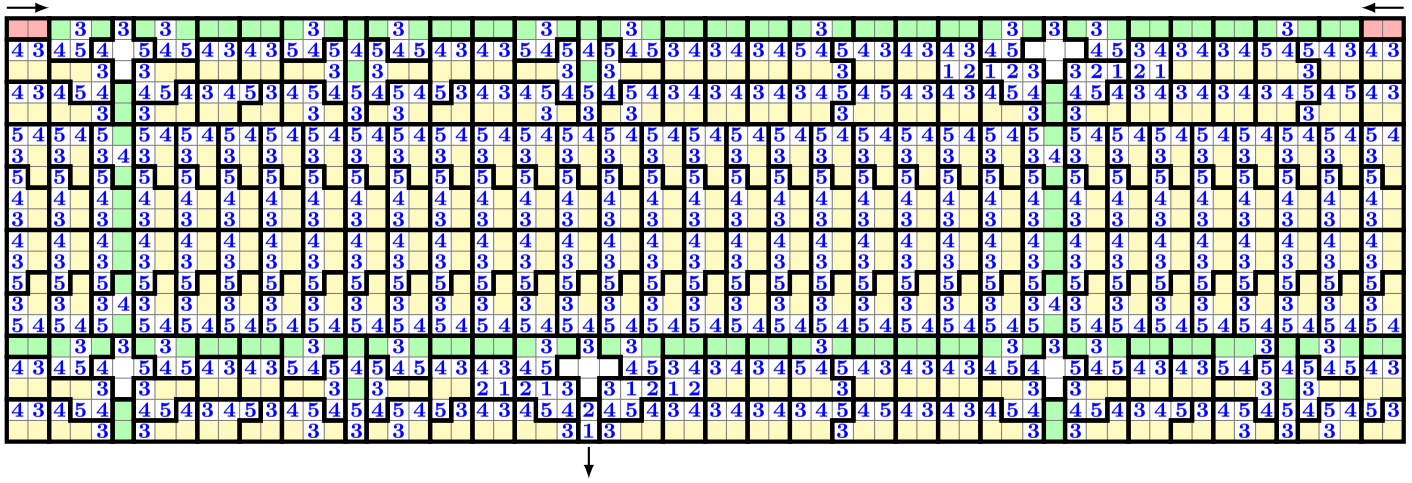
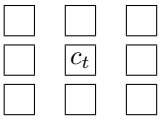


Figure 15: $(\overline{x_1} \vee x_2) \wedge x_1$ represented in Suguru. The inputs are indicated with arrows. The input for x_1 is on the left part, the input for x_2 is on the right part. To check if the formula is valid we set the output to TRUE, hence if there is a solution for the grid then the formula is valid, otherwise the solution in not satisiable.



Thus, a cell can have at most eight neighbors.

Pile-scramble shuffle. A shuffle used in our protocol is a *pile-scramble shuffle*, which was first used by Ishikawa et al. [13] and was used in other physical ZKP protocols for puzzles (e.g., Sudoku [33]). Consider that we have a sequence of ℓ piles of cards, each of which consists of the same number of face-down cards, denoted by $(p_1, p_2, \ldots, p_\ell)$ for some positive integer ℓ. Applying a pile-scramble shuffle to the sequence results in $(p_{r^{-1}(1)}, p_{r^{-1}(2)}, \ldots, p_{r^{-1}(\ell)})$ where permutation r is uniformly and randomly chosen from the symmetric group of degree ℓ. That is, it randomly permutes a sequence of piles, and nobody knows the order of the resulting sequence.

One can easily implement a pile-scramble shuffle using physical tools that can fix each pile of cards such as rubber bands and envelopes; a player (or players) randomly shuffles them until nobody traces the order of the piles.

4. ZKP protocol for Suguru

We propose a ZKP protocol for Suguru composed of two phases, the setup phase, and the verification phase.

4.1. Setup phase

The verifier V and prover P place commitments corresponding to the integers on the initial grid of a Suguru puzzle. In addition, when a region of k cells is already filled with $k-1$ cells, P and V agreed on the last cell to complete and place the commitment accordingly[7].

Subsequently, P continues to place commitments on all the remaining cells by himself according to the solution of the puzzle.

4.2. Verification phase

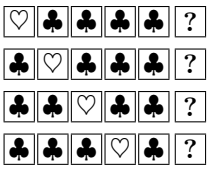
There are two verifications to ensure the number region rule and neighbor rule.

Number region rule. V wants to check that a region of k cells contains all the consecutive integers from 1 to k. The following steps are exactly the same as in the ZKP protocol for Makaro for verifying the room rule [4].

1. For every i, $1 \leq i \leq k$, V picks all cards of the i-th cell (in any ordering) to form a pile p_i. Subsequently, V attaches a numbered card $\boxed{i}$ to p_i. Thus, there are $p_1, \ldots, p_k$ piles, each of which consists of six cards.

[7]For example, in Figure 1 the upper left region can be directly completed with a 1.

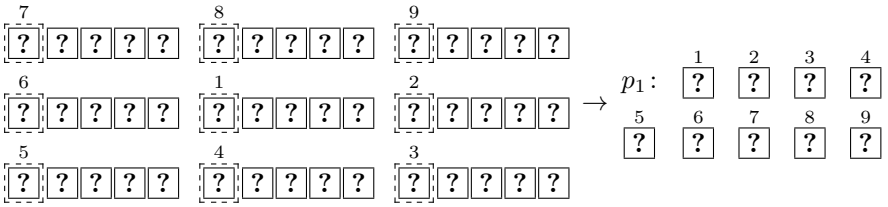
2. Apply the pile-scramble shuffle.
3. V reveals the cards of each pile except for the numbered card. The revealed output is of the form (up to a permutation in the rows), *i.e.*, all the k (opened) commitments corresponding to 1 through k should appear. For example, if $k = 4$, the revealed output should be of the following form (up to a permutation in the rows):



where the face-down cards on the right side are the numbered cards. If the revealed output is not of this form, V aborts.
4. Turn over the face-up cards and apply the pile-scramble shuffle again to the piles.
5. Reveal only the numbered cards of all piles. Because these revealed cards indicate the initial positions for each pile, V rearranges each pile back to their initial place. The revealed numbered cards can be reused for the remaining verifications.

Neighbor rule. V wants to check if a given cell has no neighbor with the same integer as the cell.[8]

1. V picks the first card of the target commitment and then picks each first card of the commitments on its neighbor cells (in any order) to form the pile p_1. The following is an example when there are eight neighbors:



2. V repeats the same operation until the pile p_5 is formed.
3. V attaches a numbered card $\boxed{i}$ to p_i. (If the target cell is the last one, V does not perform this step.)

[8] This neighbor rule can be also verified using the existing ZKP protocols for Makaro [4] and Numberlink [31]. Our proposed protocol differs with those protocols in that we simultaneously verify that a value on a given cell is different from the values on its eight neighbors using a pile-scramble shuffle. In the protocol for Makaro [4], it verifies that all adjacent cells have different values. In the protocol for Numberlink [31], it uses an extension of a pile-scramble shuffle called a "double-scramble" shuffle.

4. If an integer is written on the target cell, then go to the next step. Otherwise, apply the pile-scramble shuffle to the piles.
5. V reveals the first card of each pile, which corresponds to the target commitment. Let p_t denote the pile where a red card appears.
6. V reveals all the cards in the pile p_t except for the numbered cards. If there are two red cards in the pile, then V aborts; otherwise, V goes to the next step.
7. As Steps 4 and 5 in the previous verification, V rearranges all the cards in the piles back to their initial places. (If the target cell is the last one, V does not perform this step.)

4.3. Evaluation

If the size of the grid is represented by n, then the number of cards used in this protocol is equal to $5n+5$. Each cell must be encoded with five cards (four blacks and one red)[9] and five numbered cards are used for all the Starget cells.

5. Security proofs

We present theorems along with their proofs to show that our protocol respects the security properties, which were introduced in Section 1.

Theorem 1 (Completeness). *If P knows a solution of a Suguru grid, then it can convince V.*

Proof. Suppose that the prover P knows the solution of the Suguru grid. The prover runs with the verifier V the Setup phase (Section 4.1). We show that P can perform both verification phases without aborting.

Number region verification:. In this phase, the goal of P is to show that each region of size k contains consecutive integers from 1 to k (note that the lower bound of k is 1 and its upper bound 5). Because P places the cards accordingly with the solution, each region of size k contains the numbers 1 to k. Without loss of generality, suppose that the pile p_i corresponds to the number i with $i = 1 \dots k$. The pile p_1 is composed of the sequence (in this order):

 .

The pile p_2 is composed of the sequence (in this order):

 .

[9] We could have encoded each cell with a total of ℓ cards where ℓ is the number of cells in the region (thus, a region with two cells has its cell encoded with only two cards, a red and a black). Yet, this would lead to inconstancy in the encoding rule, which is required in the neighbor verification.

More generally, the pile p_i is a sequence of black cards where the red card is placed at position i.

Because the pile-scramble shuffle applied on Step 2 does not modify the order of the sequence, the red card of pile p_i is at position i. As $i = 1 \dots k$, all numbers from 1 to k are represented. Thus, V is convinced that the number region rule is verified by revealing the piles in Step 3.

Neighbor verification:. The goal of P is to convince V that no cell has the same number of its neighbors (there are eight neighbors as defined in Section 3). Let c_t be the target cell placed in the center of the 3×3 square. Because P placed the commitments according to the solution, there is no cell with the same value of c_t in this square. Let i be the position of the red card of c_t (with $i = 1 \dots 5$). Because no neighbor cell has the same value of c_t, there is no other red card at position i. Because each pile is composed of cards with the same indices, the pile p_i (before the shuffle) contains exactly one red card. Hence, V is convinced that the neighbor rule is verified.

Finally, because all verifications are checked, we proved that if P has the solution, then the verifications will always succeed. □

Theorem 2 (Extractability). *If P does not provide a solution of the Suguru puzzle, then it is not able to convince V.*

Proof. Suppose that P does not know a solution for the puzzle. We need to show that V will always detect it.

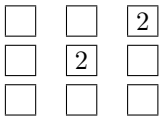
Because P cannot provide the solution, at least one of the two rules is not verified (if both can be verified, this is the solution). We can distinguish the two cases corresponding to each verification:

- The number region rule is not respected. That is, suppose w.l.o.g. that a region of size k with $k > 1$ does not contain the number 1. Hence, the sequence corresponding to this number is missing, meaning that V cannot reveal the sequence in Step 3:

   .

  Thus, V will abort the protocol and detect that P cannot provide the solution.

- The neighbor rule is not respected. Suppose that we have the following configuration:

where the blank cells have different values from 2.

We encode number 2 as:

 .

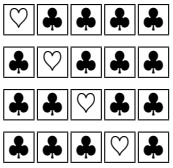
Thus, the pile p_2 corresponding to all the cards with index 2 (before the shuffle) will contain exactly two red cards. Thus, V will abort the protocol.

We proved that if P does not have the solution, then the verifications will abort in both cases meaning that P cannot convince V. □

Theorem 3 (Zero-knowledge). *V learns nothing about P's solution of the given grid G.*

Proof. We use the same proof technique as in [11]: zero-knowledge is caused by a description of an efficient *simulator* that simulates interaction between a cheating verifier and a real prover. However, the simulator does not have a solution, but it can swap cards for different ones during shuffles. The simulator acts as follows:

- During the number region verification at Step 2, the simulator swaps the piles to replace them by the sequences (up to a permutation in the rows):



- During the neighbor verification, when revealing the cards at Step 6, the simulator swaps the pile with a pile containing $k-1$ black cards and 1 red card.

The simulated and real proofs are indistinguishable; thus, V learns nothing about P's solution. □

6. Conclusion

In this study, we proposed a simple card-based physical ZKP for Suguru. Our solution is simple and efficient because it relies on only $5n + 5$ cards. We also proved that solving a Suguru grid is NP-complete.

Moreover, our long term research direction is to design physical ZKP protocols for all Nikoli's games. However, some rules of some games such as Shakashaka[10] require drawing rectangles, which is not easy to model without

[10] `http://www.nikoli.co.jp/en/puzzles/shakashaka.html`

leaking any information. Another example of a challenging game is Shikaku[11], where the rules are simple: 1) Divide the grid into rectangles with the numbers in the cells. 2) Each rectangle is to contain only one number showing the number of cells in the rectangle. However, it remains a challenging open question to design a physical ZKP for this game without revealing any information on the positions of the rectangles.

Acknowledgements.

We thank the anonymous referees, whose comments have helped us to improve the presentation of the paper. This work was supported in part by JSPS KAKENHI Grant Numbers JP17K00001, JP19J21153, and JP21K11881. This study was partially supported by the French ANR project ANR-18-CE39-0019 (MobiS5). This work has been partially supported by the French government research program "Investissements d'Avenir" through the IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25) and the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01). This work was also supported by the French ANR project DECRYPT (ANR-18-CE39-0007) and SEVERITAS (ANR-20-CE39-0009).

References

[1] Y. Abe, M. Iwamoto, K. Ohta, Efficient Private PEZ Protocols for Symmetric Functions, in: D. Hofheinz, A. Rosen (Eds.), Theory of Cryptography, vol. 11891 of *LNCS*, Springer, Cham, 372–392, 2019, https://doi.org/10.1007/978-3-030-36030-6_15.

[2] J. Balogh, J. A. Csirik, Y. Ishai, E. Kushilevitz, Private computation using a PEZ dispenser, Theor. Comput. Sci. 306 (1) (2003) 69–84, https://doi.org/10.1016/S0304-3975(03)00210-X.

[3] X. Bultel, J. Dreier, J. Dumas, P. Lafourcade, Physical Zero-Knowledge Proofs for Akari, Takuzu, Kakuro and KenKen, in: E. D. Demaine, F. Grandoni (Eds.), Fun with Algorithms, vol. 49 of *LIPIcs*, Schloss Dagstuhl, Dagstuhl, Germany, 8:1–8:20, 2016, https://doi.org/10.4230/LIPIcs.FUN.2016.8.

[4] X. Bultel, J. Dreier, J. Dumas, P. Lafourcade, D. Miyahara, T. Mizuki, A. Nagao, T. Sasaki, K. Shinagawa, H. Sone, Physical Zero-Knowledge Proof for Makaro, in: T. Izumi, P. Kuznetsov (Eds.), SSS 2018, vol. 11201 of *LNCS*, Springer, Cham, 111–125, 2018, https://doi.org/10.1007/978-3-030-03232-6_8.

[5] R. Cramer, I. Damgård, J. B. Nielsen, Multiparty Computation from Threshold Homomorphic Encryption, in: B. Pfitzmann (Ed.), Advances

[11] https://www.nikoli.co.jp/en/puzzles/shikaku.html

in Cryptology — EUROCRYPT 2001, vol. 2045 of *LNCS*, Springer, Berlin, Heidelberg, 280–300, 2001, https://doi.org/10.1007/3-540-44987-6_18.

[6] I. Damgård, S. Faust, C. Hazay, Secure Two-Party Computation with Low Communication, in: R. Cramer (Ed.), Theory of Cryptography, vol. 7194 of *LNCS*, Springer, Berlin, Heidelberg, 54–74, 2012, https://doi.org/10.1007/978-3-642-28914-9_4.

[7] J. Dumas, P. Lafourcade, D. Miyahara, T. Mizuki, T. Sasaki, H. Sone, Interactive Physical Zero-Knowledge Proof for Norinori, in: COCOON 2019, vol. 11653 of *LNCS*, Springer, Cham, 166–177, 2019, https://doi.org/10.1007/978-3-030-26176-4_14.

[8] O. Goldreich, S. Micali, A. Wigderson, How to Prove all NP-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design, in: A. M. Odlyzko (Ed.), CRYPTO 1986, vol. 263 of *LNCS*, Springer, Berlin, Heidelberg, 171–185, 1987, https://doi.org/10.1007/3-540-47721-7_11.

[9] O. Goldreich, S. Micali, A. Wigderson, Proofs that Yield Nothing But Their Validity for All Languages in NP Have Zero-Knowledge Proof Systems, J. ACM 38 (3) (1991) 691–729, https://doi.acm.org/10.1145/116825.116852.

[10] S. Goldwasser, S. Micali, C. Rackoff, The Knowledge Complexity of Interactive Proof-systems, in: STOC 1985, ACM, 291–304, 1985, https://doi.org/10.1145/22145.22178.

[11] R. Gradwohl, M. Naor, B. Pinkas, G. N. Rothblum, Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles, Theory Comput. Syst. 44 (2) (2009) 245–268, https://doi.org/10.1007/s00224-008-9119-9.

[12] M. Holzer, A. Klein, M. Kutrib, O. Ruepp, Computational Complexity of NURIKABE, Fundam. Informaticae 110 (1-4) (2011) 159–174, https://doi.org/10.3233/FI-2011-534.

[13] R. Ishikawa, E. Chida, T. Mizuki, Efficient Card-Based Protocols for Generating a Hidden Random Permutation Without Fixed Points, in: C. S. Calude, M. J. Dinneen (Eds.), UCNC 2015, vol. 9252 of *LNCS*, Springer, Cham, 215–226, 2015, https://doi.org/10.1007/978-3-319-21819-9_16.

[14] C. Iwamoto, M. Haruishi, T. Ibusuki, Herugolf and Makaro are NP-complete, in: H. Ito, S. Leonardi, L. Pagli, G. Prencipe (Eds.), Fun with Algorithms, vol. 100 of *LIPIcs*, Schloss Dagstuhl, Dagstuhl, 24:1–24:11, 2018, https://doi.org/10.4230/LIPIcs.FUN.2018.24.

[15] P. Lafourcade, D. Miyahara, T. Mizuki, L. Robert, T. Sasaki, H. Sone, How to Construct Physical Zero-Knowledge Proofs for Puzzles with a "Single Loop" Condition, Theor. Comput. Sci. https://doi.org/10.1016/j.tcs.2021.07.019.

[16] D. Lichtenstein, Planar formulae and their uses, SIAM J. Comput. 11 (2) (1982) 329–343, https://doi.org/10.1137/0211025.

[17] Z. Liu, C. Yang, Hanano Puzzle is NP-hard, Information Processing Letters 145 (2019) 6–10, https://doi.org/10.1016/j.ipl.2019.01.003.

[18] B. P. McPhail, The Complexity of Puzzles: NP-Completeness Results for Nurikabe and Minesweeper, Bachelor Thesis, The Division of Mathematics and Natural Sciences, Reed College, 2003.

[19] B. P. McPhail, Light Up is NP-complete, https://www.researchgate.net/publication/249927572_Light_Up_is_NP-complete, 2005.

[20] D. Miyahara, Y. Hayashi, T. Mizuki, H. Sone, Practical card-based implementations of Yao's millionaire protocol, Theor. Comput. Sci. 803 (2020) 207–221, https://doi.org/10.1016/j.tcs.2019.11.005.

[21] D. Miyahara, L. Robert, P. Lafourcade, S. Takeshige, T. Mizuki, K. Shinagawa, A. Nagao, H. Sone, Card-Based ZKP Protocols for Takuzu and Juosan, in: M. Farach-Colton, G. Prencipe, R. Uehara (Eds.), Fun with Algorithms, LIPIcs, Schloss Dagstuhl, Dagstuhl, 20:1–20:21, 2020, https://doi.org/10.4230/LIPIcs.FUN.2021.20.

[22] D. Miyahara, T. Sasaki, T. Mizuki, H. Sone, Card-Based Physical Zero-Knowledge Proof for Kakuro, IEICE Trans. Fundamentals E102.A (9) (2019) 1072–1078, https://doi.org/10.1587/transfun.E102.A.1072.

[23] S. Murata, D. Miyahara, T. Mizuki, H. Sone, Public-PEZ Cryptography, in: W. Susilo, R. H. Deng, F. Guo, Y. Li, R. Intan (Eds.), Information Security, vol. 12472 of *LNCS*, Springer, Cham, 59–74, 2020, https://doi.org/10.1007/978-3-030-62974-8_4.

[24] T. Nakai, Y. Misawa, Y. Tokushige, M. Iwamoto, K. Ohta, How to Solve Millionaires' Problem with Two Kinds of Cards, New Gener. Comput. 39 (1) (2021) 73–96, https://doi.org/10.1007/s00354-020-00118-8.

[25] H. Ono, Y. Manabe, Efficient Card-Based Cryptographic Protocols for the Millionaires' Problem Using Private Input Operations, in: Asia Joint Conference on Information Security (AsiaJCIS), 23–28, 2018, https://doi.org/10.1109/AsiaJCIS.2018.00013.

[26] H. Ono, Y. Manabe, Card-Based Cryptographic Protocols with the Minimum Number of Rounds Using Private Operations, in: C. Pérez-Solà, G. Navarro-Arribas, A. Biryukov, J. Garcia-Alfaro (Eds.), Data Privacy Management, Cryptocurrencies and Blockchain Technology, vol. 11737

of *LNCS*, Springer, Cham, 156–173, 2019, https://doi.org/10.1007/978-3-030-31500-9_10.

[27] H. Ono, Y. Manabe, Card-Based Cryptographic Logical Computations Using Private Operations, New Gener. Comput. 39 (1) (2021) 19–40, https://doi.org/10.1007/s00354-020-00113-z.

[28] J. Quisquater, M. Quisquater, M. Quisquater, M. Quisquater, L. C. Guillou, M. A. Guillou, G. Guillou, A. Guillou, G. Guillou, S. Guillou, T. A. Berson, How to Explain Zero-Knowledge Protocols to Your Children, in: G. Brassard (Ed.), Advances in Cryptology — CRYPTO' 89, vol. 435 of *LNCS*, Springer, New York, 628–631, 1990, https://doi.org/10.1007/0-387-34805-0_60.

[29] L. Robert, D. Miyahara, P. Lafourcade, T. Mizuki, Physical Zero-Knowledge Proof for Suguru Puzzle, in: S. Devismes, N. Mittal (Eds.), Stabilization, Safety, and Security of Distributed Systems, vol. 12514 of *LNCS*, Springer, Cham, 235–247, 2020, https://doi.org/10.1007/978-3-030-64348-5_19.

[30] C. Romero-Tris, J. Castellà-Roca, A. Viejo, Multi-party Private Web Search with Untrusted Partners, in: M. Rajarajan, F. Piper, H. Wang, G. Kesidis (Eds.), Security and Privacy in Communication Networks, vol. 96 of *LNICST*, Springer, Berlin, Heidelberg, 261–280, 2012, https://doi.org/10.1007/978-3-642-31909-9_15.

[31] S. Ruangwises, T. Itoh, Physical Zero-Knowledge Proof for Numberlink Puzzle and k Vertex-Disjoint Paths Problem, New Gener. Comput. 39 (1) (2021) 3–17, https://doi.org/10.1007/s00354-020-00114-y.

[32] S. Ruangwises, T. Itoh, Physical Zero-Knowledge Proof for Ripple Effect, in: S. Hong, S. Nandy, R. Uehara (Eds.), WALCOM: Algorithms and Computation, vol. 11737 of *LNCS*, Springer, Cham, 296–307, 2021, https://doi.org/10.1007/978-3-030-68211-8_24.

[33] T. Sasaki, D. Miyahara, T. Mizuki, H. Sone, Efficient card-based zero-knowledge proof for Sudoku, Theor. Comput. Sci. 839 (2020) 135–142, https://doi.org/10.1016/j.tcs.2020.05.036.

[34] A. Shamir, IP = PSPACE, J. ACM 39 (4) (1992) 869–877, https://doi.org/10.1145/146585.146609.

[35] K. Shinagawa, K. Nuida, A single shuffle is enough for secure card-based computation of any Boolean circuit, Discrete Appl. Math. 289 (2021) 248–261, https://doi.org/10.1016/j.dam.2020.10.013.

[36] K. Takashima, Y. Abe, T. Sasaki, D. Miyahara, K. Shinagawa, T. Mizuki, H. Sone, Card-based protocols for secure ranking computations, Theor. Comput. Sci. 845 (2020) 122–135, https://doi.org/10.1016/j.tcs.2020.09.008.