Zero-Knowledge Proof of Knowledge for Peg Solitaire

Xavier Bultel
INSA Centre Val de Loire, Laboratoire d'informatique fondamentale d'Orléans, Bourges, France

Abstract

Peg solitaire is a very popular traditional single-player board game, known to be NP-complete. In this paper, we present a zero-knowledge proof of knowledge for solutions of peg solitaire instances. Our proof is straightforward, in the sense that it does not use any reduction to another NP-complete problem, and uses the standard design of sigma protocols. Our construction relies on cryptographic commitments, which can be replaced by envelopes to make the protocol physical. As a side contribution, we introduce the notion of isomorphisms for peg solitaire, which is the key tool of our protocol.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Zero-Knowledge Proof, Peg Solitaire

Digital Object Identifier 10.4230/LIPIcs.FUN.2022.9

1 Introduction

Zero-knowledge proofs are fascinating protocols introduced by Goldreich, Micali, and Rackoff [8], in which a *prover*, knowing a secret needed to verify some property, tries to convince a verifier that the property actually holds, but without revealing anything about the secret. Although these surprising features seem difficult to achieve, these protocols are often elegant and very simple to understand. For this reason, zero-knowledge proofs are usually considered as a beautiful concept.

In [7], Goldreich, Micali, and Wigderson present a protocol for proving the knowledge of a graph 3-coloring, without revealing anything about the coloring. Such a protocol is said to be a *zero-knowledge proof of knowledge*. This protocol is amazingly simple: the prover randomly permutes the three colors and commits the new color of each vertex, the verifier chooses one of the edges of the graph, the prover reveals the color of its two endpoints, and the verifier checks that these two colors are different. On the one hand, the verifier learns nothing else than the fact that the two endpoints have different (random) colors. On the other hand, if the prover does not know any 3-coloring, then at least one edge has two endpoints of the same color. In this case, the prover succeeds its proof with probability of at most $(n-1)/n$, where n is the number of edges. By repeating the protocols λ times, its probability of success falls to $((n-1)/n)^\lambda$. By adjusting the parameter λ, the probability of deceiving the verifier can be reduced as much as desired. This protocol can be used physically (*i.e.* without computer and without cryptography) by replacing the commitments with paper envelopes.

This construction allows at the same time to show the existence of a zero-knowledge proof of knowledge for any problem in NP: by reducing the required problem to the graph 3-coloring, the prover can turn the instance of the problem into an instance of the graph 3-coloring, and can prove its knowledge of this 3-coloring. However, using this generic method results in heavy and unclear proof protocols, and it is often better to design tailor-made proofs for specific problems in NP that are understandable, elegant and efficient.

If building tailor-made zero-knowledge proofs of knowledge for NP problems is fun in itself, building such proofs for fun problems is even more fun. Therefore, many proofs of knowledge for logic puzzles have been proposed, either in a cryptographic or a physical setting. For

instance, cryptographic and physical zero-knowledge proofs of knowledge for sudoku puzzles are presented in [9] and [16], physical proofs for various logical grid puzzles (namely akari, takuzu, kakuro and ken-ken) are presented in [4], and cryptographic proofs problems based on the Rubik's cube are presented in [19]. Some works use physical properties of playing cards to optimize the design of the physical proofs [13, 16]. As a fun application, such proofs can be used as authentication or signature protocols, where authentication amounts to proving the knowledge of a secret solution for a public logic game instance [19].

Peg solitaire is a traditional board game for one player known all over the world. This game was already known in the 17^{th} century, and is probably older. Peg solitaire rules are very simple but the game is deeply complex, which makes it an exciting research topic for mathematicians and computer scientists, as shown by the many scientific papers that have already been published about it [1, 2, 10, 11, 12, 15, 18]

The game is played on a board with holes that can contain pegs. The player can jump a peg over an adjacent peg (which is removed from the board) into a hole. The goal is to reach a given winning position from a given initial position. Peg solitaire is proven to be NP-complete [10]. A weaker variant of the game, where the goal is to remove all the pegs but one, was previously proven to be NP-complete in [18]. In [15], authors show that the game is no longer NP-complete when one of the dimensions of the board is fixed.

In this paper, we give the first tailor-made cryptographic zero-knowledge proof of knowledge for solutions of peg solitaire, in the sense that our protocol does not rely on a reduction to another problem but uses the specific properties of the peg solitaire. While most logic games used to build zero-knowledge proofs consist of filling in a grid with a pen within certain constraints, peg solitaire has a different mechanism, since its solution is a series of successive moves on a board. The method that we use seems generic and should be applicable to other similar games: at each proof round, the prover shows to the verifier that one of its moves is legal.

Our construction

Because of its structure, a peg solitaire board can be formalized as a graph (the link between the graphs and the peg solitaire is quite natural and has already been studied in [2]). This representation allows us to extend the notion of graph isomorphisms to define the notion of peg solitaire isomorphisms. Loosely speaking, two peg solitaires are isomorphic when there is a bijection that allows to pass from one to the other preserving the existence and the general structure of their solutions. Considering two isomorphic peg solitaires, we give an efficient method to transform a solution of one into a solution of the other using the isomorphism. Our definition of peg solitaire isomorphism is generic and could be of independent interest.

Our proof protocol uses the standard method introduced in [7] with the proof for 3-coloring: the prover commits its solution, the verifier challenges the prover to show that one of the constraints of the problem holds in the solution, and the prover shows this by decommitting one part of the solution. Obviously, the revealed part should not leak anything else than the respect of the constraint.

More precisely, the prover first chooses an isomorphism of peg solitaire at random, and computes the image of the peg solitaire instance and its solution by this isomorphism. This step *randomizes* the structure of the solution. The prover then commits each part of this randomized peg solitaire and its solution. The verifier has two ways to challenge the prover:

- Either it requests the prover to reveal the isomorphism. In this case the prover also decommits the isomorphic peg solitaire (but not its solution). The verifier then checks that the peg solitaire has been correctly randomized, but it learns nothing about the solution.

- Or it requests the prover to reveal the i^{th} move of the randomized solution. In this case the prover decommits the corresponding part of the randomised solution, and the verifier checks that the move is legal (*i.e.* it is a jump from a peg over an adjacent peg landing in a hole). Since this move is isolated from the others and takes place on a randomized board, the verifier does not learn anything else about the solution.

If the prover knows how to answer each challenge, then it knows each move of the solution for the isomorphic peg solitaire instance, and it knows the isomorphism allowing to pass from this solution to that of the initial peg solitaire instance. By contrapositive, if the prover is not able to answer all the challenges, then it does not know a solution. The probability that the prover deceives the verifier is at most $(S-1)/S$, where S is the number of steps in the solution. By repeating the protocol several times, this probability becomes as small as desired. Note that this protocol can also be turned into physical proof by replacing each commitment with a paper envelope.

2 Cryptographic background

In this section, we recall the definitions of negligible functions, zero-knowledge proofs of knowledge, sigma protocols, and commitments.

We use the notion of *negligible function* to formalize the concept of *unrealistic event*. A function is negligible when it tends to zero faster than the inverse of any polynomial. We consider that an attack is not feasible in practice if its probability is negligible in some security parameter.

▶ **Definition 1.** *A positive function* $\epsilon : \mathbb{N} \to \mathbb{R}$ *is said to be* negligible *if for any positive polynomial* t*, there exists an integer* n *such that for all integer* $x > n$*,* $|\epsilon(x)| < \frac{1}{t(x)}$.

A *zero-knowledge proof of knowledge* [3] is a protocol where two parties interact, a *prover* and a *verifier*. The prover knows a secret value (in our case, the solution of a peg solitaire) and tries to convince the verifier of this knowledge, without leaking any other information about the solution.

A zero-knowledge proof of knowledge has the three following properties:

Completness: If the prover actually knows the secret and runs the protocol honestly, then the proof is accepted.

Validity: If the prover does not know the secret, then the probability that its proof is accepted in a reasonable time is negligible. This probability is called the *knowledge error*. More precisely, assuming the existence of a prover able to perform an accepted proof with a reasonable probability, the validity ensures that the secret of that prover can be extracted in a similar running time.

Zero-knowledge: The verifier learns nothing about the secret of the prover. More precisely, the verifier is able to simulate its interaction with the prover in such a way that no efficient algorithm can distinguish the simulated transcripts from the real transcripts with non-negligible probability.

The protocol that we present in this paper is a *sigma protocol* [5], *i.e.* it is a repetition of the three following interactions: the prover sends a commitment, the verifier sends a challenge, and the prover sends a response. Due to their structure, validity of such protocols can be proven using the definition of the *n-Special-soundness*. This property holds when it is possible to extract the secret from n transcripts containing different challenges but sharing the same commitment. A n-special-sound sigma protocol is valid with a knowledge error of $(n-1)/n$ [5]. By repeating the protocol λ times, the knowledge error falls to $((n-1)/n)^{\lambda}$, which is negligible in λ.

▶ **Definition 2** (Sigma protocol [3, 5]). *Let R_L be a relation for a language L in NP, we say that an element x is a* witness *for the* instance $y \in L$ *if* $(x, y) \in R_L$. *Let λ be a security parameter; in what follows,* p.p.t algorithm *means* probabilistic polynomial-time algorithm in λ. *Let us consider two p.p.t algorithms $\mathcal{P}$ (the Prover) knowing a witness x and $\mathcal{V}$ (the verifier) knowing an instance y interacting in a proof protocol $\langle \mathcal{P}(x), \mathcal{V}(y) \rangle$ using the following pattern: $\mathcal{P}$ sends a* commitment, *$\mathcal{V}$ sends a* challenge, *$\mathcal{P}$ sends a* response, *after which $\mathcal{V}$ accepts or rejects the proof. This proof protocol is said to be a* sigma-protocol *for R_L. We denote the set of the accepted transcripts for any instance y by $\mathsf{Acc}(y)$. Moreover, a sigma protocol can have the following properties:*

Completeness: *For every $(x, y) \in R_L$:*

$$\mathsf{Pr}[\langle \mathcal{P}(x), \mathcal{V}(y) \rangle \in \mathsf{Acc}(y)] = 1.$$

n-Special-soundness: *For any $y \in L$, there exists an efficient algorithm $\mathcal{E}$ (polynomial in $|y|$) such that for any set of n valid transcripts $\tau \in \mathsf{Acc}(y)^n$ sharing the same commitment but having n different challenges from each other, we have:*

$$\mathsf{Pr}[x \leftarrow \mathcal{E}(\tau, y) : (x, y) \in R_L] = 1.$$

(Computational) Zero-knowledge: *For every $(x, y) \in R_L$ and every p.p.t verifier $\mathcal{V}^*$ there exists a probabilistic p.p.t algorithm $\mathcal{S}$ (called* simulator*) such that for every p.p.t algorithm $\mathcal{D}$, the following is negligible in λ:*

$$|\mathsf{Pr}[\mathsf{tr} \leftarrow \langle \mathcal{P}(x), \mathcal{V}^*(y) \rangle; b \leftarrow \mathcal{D}(\mathsf{tr}) : b = 1] - \mathsf{Pr}[\mathsf{tr} \leftarrow \mathcal{S}(y); b \leftarrow \mathcal{D}(\mathsf{tr}) : b = 1]|.$$

A sigma protocol which is complete, special-sound, and zero-knowledge is said to be a zero-knowledge proof of knowledge.

A *cryptographic commitment* scheme [6, 7] $C = (\mathsf{Gen}, \mathsf{Commit})$ is a pair of algorithms that allows a user to commit a value with a secret key. More precisely, $\mathsf{Gen}(1^\lambda)$ generates a commitment key sk from a security parameter λ, and $\mathsf{Commit}(\mathsf{sk}, m)$ generates a commitment c from a key sk and a value m. To open the commitment c, the user reveals the committed value m and the key sk. The validity of the opened commitment is verified if $c = \mathsf{Commit}(\mathsf{sk}, m)$. Such a scheme should verify two properties:

Biding: The user is constrained to open the value that it actually committed. In other words, the user cannot open a commitment in two different ways with non-negligible probability. If this probability is zero, we say that the commitment has the *perfect* binding property.

Hiding: Without the key, a commitment reveals nothing about the committed value. In other words, a user cannot distinguish which value is committed out of two chosen values in a given commitment with non-negligible probability.

▶ **Definition 3** (Commitment Scheme [6, 7]). *A* commitment scheme $C = (\mathsf{Gen}, \mathsf{Commit})$ *is a pair of algorithms defined as follows:*

- $\mathsf{Gen}(1^\lambda)$ *generates a commitment key* sk *from a security parameter λ.*
- $\mathsf{Commit}(\mathsf{sk}, m)$ *generates a commitment c from a key and a message.*

A commitment scheme C has the binding *(resp.* perfect biding*) security property if for any p.p.t algorithm $\mathcal{A}$, the following is negligible in λ (resp. null), where $\boldsymbol{Exp}_{C,\mathcal{A}}^{\mathsf{Binding}}(\lambda)$ denotes the experiment given in Figure 1 (right side):*

$$\mathsf{Pr}\left[1 \leftarrow \boldsymbol{Exp}_{C,\mathcal{A}}^{\mathsf{Binding}}(\lambda)\right].$$

Experiment $\mathbf{Exp}^{\mathsf{Binding}}_{C,\mathcal{A}}(\lambda)$:
$(\mathsf{sk}_0, \mathsf{sk}_1, m_0, m_1) \leftarrow \mathcal{A}(1^\lambda)$
if $\mathsf{Commit}(\mathsf{sk}_0, \mathsf{m}_0) = \mathsf{Commit}(\mathsf{sk}_1, \mathsf{m}_1)$
then return $m_0 \neq m_1$
else return 0

Experiment $\mathbf{Exp}^{\mathsf{Hiding}}_{C,\mathcal{A},b}(\lambda)$:
$\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$
$(\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{sk})$
$c_b \leftarrow \mathsf{Commit}(\mathsf{sk}, \mathsf{m}_b)$
$b' \leftarrow \mathcal{A}_2(c_b, \mathsf{st})$
return $b = b'$

Figure 1 The *binding* and the *hiding* experiments.

A commitment scheme has the hiding *security property if for any two-party p.p.t algorithms* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the following is negligible in* λ*, where* $\boldsymbol{Exp}^{\mathsf{Hiding}}_{C,\mathcal{A}}(\lambda)$ *denotes the experiment given in Figure 1 (left side):*

$$\left|\mathsf{Pr}\left[1 \leftarrow \boldsymbol{Exp}^{\mathsf{Hiding}}_{C,\mathcal{A},1}(\lambda)\right] - \mathsf{Pr}\left[0 \leftarrow \boldsymbol{Exp}^{\mathsf{Hiding}}_{C,\mathcal{A},0}(\lambda)\right]\right|.$$

In [7], the authors show how to construct a commitment scheme that has both hiding and perfect binding properties under the hypothesis of the existence of an ideal hash function, and use it as a key tool to construct zero-knowledge proofs of knowledge for NP problems. A perfect hiding commitment scheme is given in [14]. Note that a commitment scheme cannot be both perfect hiding and perfect binding.

3 The game

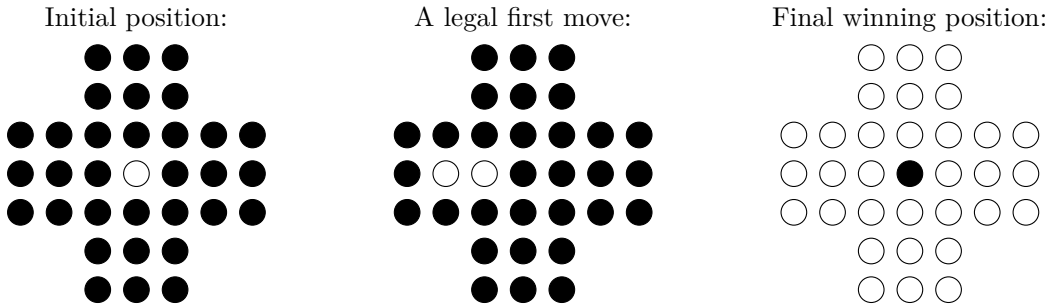

Figure 2 The english peg solitaire.

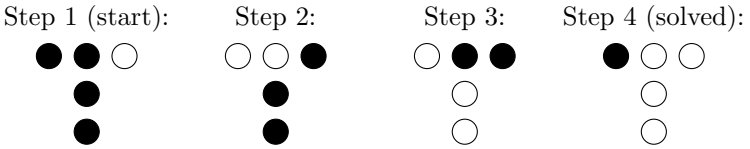

Figure 3 The solution of the tee peg solitaire.

Peg solitaire is a board game for one player, which is played on a board containing holes where pegs can be inserted. Each hole can hold at most one peg. At the beginning of the game, the pegs are placed in a given initial position, and the goal is to move them in order to reach a given winning position. The moves are jumps: if two pegs and an empty hole are

aligned and adjacent, the first peg can jump over the second one to land in the empty hole. The jumped peg is removed from the game. Since at each move a single peg is removed from the game, the number of moves to reach the wining position is equal to the number of pegs in the initial position less the number of pegs in the winning one.

In Figure 2 we present the English peg solitaire, which is one of the most popular variants of this game. The black circles are holes containing pegs, and the white circles are empty holes. In this variant, the holes are placed orthogonaly and form a cross. In the initial position, only the central hole is empty. The goal is to remove all the pegs except one, which must be moved into the center hole. Note that the possibilities of boards are endless, and are not limited to orthogonal boards.

Throughout this paper, we will use a simple example of peg solitaire to illustrate our definitions: the tee peg solitaire. The initial board contains 5 holes that form a tee, with the rightmost hole being the only empty hole. The winning position contains only one peg in the leftmost hole. In Figure 3, we show how to solve this peg solitaire, *i.e.* we show successive moves that allow to reach the winning position from the initial one.

4 Formalism

In this section, we present how we formalize the peg solitaires. In a first step, we represent the state of the game board by a graph, where each hole corresponds to a vertex labeled by an index. The value of a vertex is 1 if the corresponding hole contains a peg, 0 otherwise. The vertices corresponding to adjacent holes on the board are linked by an edge labeled by a direction (horizontal or vertical). Note that this representation can be generalized for a number of directions greater than two (for example for hexagonal boards).

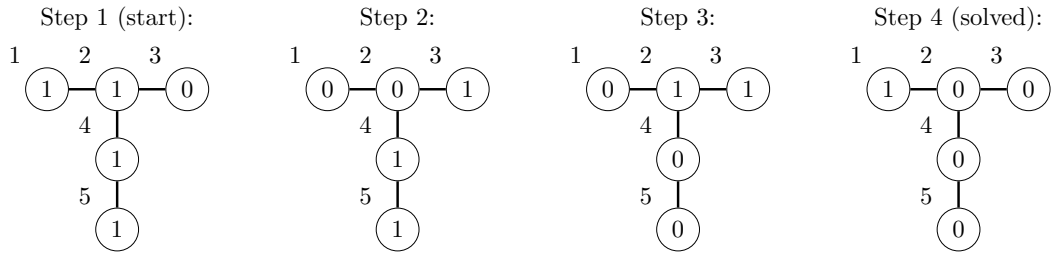

Figure 4 The graph representation of the steps of the tee peg solitaire solution.

For instance, the steps of the solution of the tee peg solitaire can be represented as in Figure 4, where the direction of each edge implicitly corresponds to its actual direction on the diagram.

A peg solitaire instance is defined by the shape of the board, the position of the pegs at the beginning of the game, and the position of the pegs that must be reached to win. We can therefore represent any instance by the graph of its initial position and the graph of its winning position. Note that only the values of the vertices differ between these two graphs, so we can give only the structure of the graph and the values of the vertices in the initial position and in the winning position.

Finally, we can encode an instance of peg solitaire by giving a set of triplets, containing the ordered combinations of 3 indexes of vertices that are adjacent and aligned in the graph (*i.e.* the triplets (a, b, c) such that (a, b) and (b, c) are edges labeled by the same direction). This is sufficient to rebuild the entire structure of the graph. For instance, the graph of the

tee peg solitaire and the graph of the English peg solitaire are encoded by 2 and 38 triplets of vertices respectively. The encoding must also contain two vectors f (for *first*) and l (for *last*), containing the values of the vertices in the initial position and in the winning position respectively, ordered by the indexes of the vertices. This leads us to the following definition, where N denotes the number of vertices, T the number of triplets, and S the number of steps in the solution.

▶ **Definition 4** (Peg solitaire instance). *Let N, S, and T be three positive integers. A (N, T, S)*-peg solitaire instance *is a tuple (t, f, l) where*

- *$t = (t_i)_{1 \leq i \leq T}$ is a vector of T triplets of integers, where:*
  - *Each t_i is in $[\![1, N]\!]^3$.*
  - *For each i, parsing t_i as $(t_{i,1}, t_{i,2}, t_{i,3})$, we have $t_{i,1} \neq t_{i,2}$, $t_{i,2} \neq t_{i,3}$ and $t_{i,1} \neq t_{i,3}$ (the values in the triplets are different).*
  - *For each $i \neq i'$, parsing t_i as $(t_{i,1}, t_{i,2}, t_{i,3})$ and $t_{i'}$ as $(t_{i',1}, t_{i',2}, t_{i',3})$, we have $\{t_{i,1}, t_{i,2}, t_{i,3}\} \neq \{t_{i',1}, t_{i',2}, t_{i',3}\}$ (each triplet contains different values).*
- *$f = (f_i)_{1 \leq i \leq N}$ and $l = (l_i)_{1 \leq i \leq N}$ are vectors of N bits and $S = 1 + \sum_{i=1}^{N} (f_i - l_i)$ (S is the difference between the number of 1 in f and l plus one, because at each step, a peg is removed).*

Note that our definition is generic, and accepts instances that could not be represented by graphs.

▶ **Example 5** (Tee peg solitaire instance). Using the previous definition, the tee peg solitaire is encoded by a $(5, 2, 4)$-peg solitaire instance (t, f, l) defined as follows:

- $t = ((1, 2, 3), (2, 4, 5))$.
- $f = (1, 1, 0, 1, 1)$ and $l = (1, 0, 0, 0, 0)$.

We now show how to encode a solution and how to verify its validity for a given peg solitaire instance. A solution is simply the set of vectors representing the values of the vertices at each step of the game. Thus, the first vector of the solution must be f, and the last vector must be l (*i.e.* the sequence of vectors must show how to go from the position of f to the position of l).

To be considered as valid, the solution must match steps that respect the rules of the game. To formalize this, let us notice two properties:

- At each step, there are exactly three aligned holes whose state changes: the hole containing the jumping peg, the hole containing the jumped peg, and the hole receiving the jumping peg.
- The state of these three aligned holes is (●●○) or (○●●) before the move, and (○ ○ ●) or (● ○ ○) respectively after the move. The direction (horizontal or vertical) does not matter here, since both cases are symmetric.

Thus, we must verify that exactly three bits are different between each vector of the solution, and that these three bits are the values of three vertices indexed in one of the triplets of the instance. Moreover, we must verify that :

- either the two first vertices of the triplet have the value 1 and the last one has the value 0 in the first vector,
- or the first vertex of the triplet has the value 0 and the two last ones have the value 1 in the first vector.

Finally, we obtain the following definition.

▶ **Definition 6** (Solution for a peg solitaire instance)**.** *Let (t, f, l) be a (N, T, S)-peg solitaire instance. A* solution *for this instance is a binary matrix $Z = (Z[i, j])_{1 \leq i \leq S; 1 \leq j \leq N}$. Throughout this paper, by abuse of notation, for any solution Z and any index i, we will denote the i^{th} line of a solution by $Z[i]$,* i.e *$Z[i] = (Z[i, j])_{1 \leq j \leq N}$.*

A solution is said to be valid *if $f = Z[1]$, $l = Z[S]$, and for any $i \in [\![1, S-1]\!]$, there exists $j \in [\![1, T]\!]$ such that, parsing t_j as $(t_{j,1}, t_{j,2}, t_{j,3})$:*

- *For any $k \in [\![1, N]\!]$, $(k \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i, k] \neq Z[i + 1, k])$ (exactly three bits, corresponding to a triplet of vertices, are different).*
- *$Z[i, t_{j,1}] \neq Z[i, t_{j,3}]$ and $Z[i, t_{j,2}] = 1$ (the three vertices are in a state where the jump is possible).*

▶ **Example 7** (Solution for the tee peg solitaire)**.** Using the previous definition, the solution for the tee peg solitaire instance given in Example 5 is encoded by the following matrix:

$$Z = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We now define the notion of peg solitaire isomorphism. Loosely speaking, two peg solitaires are isomorphic when one can be obtained by permuting the vertices, permuting the triplets, and reversing the order of the triplets of the other.

▶ **Definition 8** (Peg solitaire isomorphism)**.** *A (N, T, S)-peg solitaire* ismorphism *is a triplet of bijections $\xi = (\phi, \chi, \psi)$ defined as follows:*

- *$\phi : [\![1, N]\!] \to [\![1, N]\!]$,*
- *$\chi : [\![1, T]\!] \to [\![1, T]\!]$, and*
- *$\psi : \{1, 3\} \to \{1, 3\}$.*

For any (N, T, S)-peg solitaire instance $\Sigma = (t, f, l)$, we define the (N, T, S)-peg solitaire instance $\Sigma' = (t', f', l')$ isomorphic *to Σ by ξ as follows:*

- *for any $i \in [\![1, T]\!]$, $t'_{\chi(i)} = (\phi(t_{i,\psi(1)}), \phi(t_{i,2}), \phi(t_{i,\psi(3)}))$, and*
- *for any $i \in [\![1, N]\!]$, $f'_i = f_{\phi^{-1}(i)}$ and $l'_i = l_{\phi^{-1}(i)}$.*

▶ Remark 9. A more generic definition of peg solitaire isomorphism can be obtained by choosing a different function ψ for each triplet t_i. However, as this property is not required for our purpose and would make the notations more cumbersome, we do not mention it explicitly in our definition.

An isomorphism can be used to transform a solution for a peg solitaire instance into a solution for its isomorphic instance. We show this result in the following theorem.

▶ **Theorem 10.** *Let $\Sigma = (t, f, l)$ be a (N, T, S)-peg solitaire instance, $Z = (Z[i, j])_{1 \leq i \leq S; 1 \leq j \leq N}$ be a solution for Σ, $\xi = (\phi, \chi, \psi)$ be a (N, T, S)-peg solitaire isomorphism, and $\Sigma' = (t', f', l')$ be the (N, T, S)-peg solitaire instance isomorphic to Σ by ξ. Let $Z' = (Z'[i, j])_{1 \leq i \leq S; 1 \leq j \leq N}$ be a solution for Σ' such that, for any $i \in [\![1, S]\!]$ and $j \in [\![1, N]\!]$, $Z'[i, j] = Z[i, \phi^{-1}(j)]$. We have that Z' is a valid solution for Σ' if and only if Z is a valid solution for Σ.*

Proof. Throughout this proof, we use the same notation as in Definition 4, Definition 6 and Definition 8. We first prove the three following claims:

▷ Claim 11. $f = Z[1] \Leftrightarrow f' = Z'[1]$ and $l = Z[S] \Leftrightarrow l' = Z'[S]$.

Proof. For any $j \in [\![1, N]\!]$, $f'_j = f_{\phi^{-1}(j)}$ and $Z'[1, j] = Z[1, \phi^{-1}(j)]$ and $f_{\phi^{-1}(j)} = Z[1, \phi^{-1}(j)]$, so:

$$f = Z[1] \Leftrightarrow f' = Z'[1].$$

Moreover, $l'_j = l_{\phi^{-1}(j)}$ and $Z'[S, j] = Z[S, \phi^{-1}(j)]$ and $l_{\phi^{-1}(j)} = Z[S, \phi^{-1}(j)]$, so:

$$l = Z[S] \Leftrightarrow l' = Z'[S].$$ ◁

▷ Claim 12. for any $i \in [\![1, S-1]\!]$, any $j \in [\![1, T]\!]$ and any $k \in [\![1, N]\!]$:

$$\begin{aligned}&(k \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i,k] \neq Z[i+1,k])\\ \Leftrightarrow &(k \in \{t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3}\} \Leftrightarrow Z'[i,k] \neq Z'[i+1,k])\end{aligned}$$

Proof. For any $i \in [\![1, S-1]\!]$, any $j \in [\![1, T]\!]$ and any $k \in [\![1, N]\!]$:

$$\begin{aligned}&\big(\ k \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i,k] \neq Z[i+1,k]\ \big) && (1)\\ \Leftrightarrow &\big(\ \phi^{-1}(k) \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i,\phi^{-1}(k)] \neq Z[i+1,\phi^{-1}(k)]\ \big) && (2)\\ \Leftrightarrow &\big(\ k \in \{\phi(t_{j,1}), \phi(t_{j,2}), \phi(t_{j,3})\} \Leftrightarrow Z[i,\phi^{-1}(k)] \neq Z[i+1,\phi^{-1}(k)]\ \big) && (3)\\ \Leftrightarrow &\big(\ k \in \{\phi(t_{j,1}), \phi(t_{j,2}), \phi(t_{j,3})\} \Leftrightarrow Z'[i,k] \neq Z'[i+1,k]\ \big) && (4)\\ \Leftrightarrow &\big(\ k \in \{\phi(t_{j,\psi(1)}), \phi(t_{j,2}), \phi(t_{j,\psi(3)})\} \Leftrightarrow Z'[i,k] \neq Z'[i+1,k]\ \big) && (5)\\ \Leftrightarrow &\Big(\ k \in \{t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3}\} \Leftrightarrow Z'[i,k] \neq Z'[i+1,k]\ \Big) && (6)\end{aligned}$$

Equivalency (2) holds because ϕ is bijective, so $\{\phi^{-1}(k)\}_{k \in [\![1,N]\!]} = [\![1, N]\!]$. Equivalency (3) holds because $(\phi^{-1}(k) \in \{t_{j,1}, t_{j,2}, t_{j,3}\}) \Leftrightarrow (k \in \{\phi(t_{j,1}), \phi(t_{j,2}), \phi(t_{j,3})\})$. Equivalency (4) holds because by definition of Z', for all i and k, $Z[i, \phi^{-1}(k)] = Z'[i, k]$. Equivalency (5) holds because $(\psi(1), \psi(3)) = (1, 3)$ or $(3, 1)$, which implies $\{\phi(t_{j,\psi(1)}), \phi(t_{j,2}), \phi(t_{j,\psi(3)})\} = \{\phi(t_{j,1}), \phi(t_{j,2}), \phi(t_{j,3})\}$. Finally, Equivalency (6) holds because by definition of Σ' we have that $(\phi(t_{j,\psi(1)}), \phi(t_{j,2}), \phi(t_{j,\psi(3)})) = (t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3})$. ◁

▷ Claim 13. For any $i \in [\![1, S-1]\!]$, and any $j \in [\![1, T]\!]$:

$$(Z[i, t_{j,1}] \neq Z[i, t_{j,3}] \text{ and } Z[i, t_{j,2}] = 1) \Leftrightarrow (Z'[i, t'_{\chi(j),1}] \neq Z'[i, t'_{\chi(j),3}] \text{ and } Z'[i, t'_{\chi(j),2}] = 1)$$

Proof. For any $i \in [\![1, S-1]\!]$, and any $j \in [\![1, T]\!]$:

$$\begin{aligned}&\big(\ Z[i, t_{j,1}] \neq Z[i, t_{j,3}] \text{ and } Z[i, t_{j,2}] = 1\ \big) && (7)\\ \Leftrightarrow &\big(\ Z'[i, \phi(t_{j,1})] \neq Z'[i, \phi(t_{j,3})] \text{ and } Z'[i, \phi(t_{j,2})] = 1\ \big) && (8)\\ \Leftrightarrow &\big(\ Z'[i, \phi(t_{j,\psi(1)})] \neq Z'[i, \phi(t_{j,\psi(3)})] \text{ and } Z'[i, \phi(t_{j,2})] = 1\ \big) && (9)\\ \Leftrightarrow &\Big(\ Z'[i, t'_{\chi(j),1}] \neq Z'[i, t'_{\chi(j),3}] \text{ and } Z'[i, t'_{\chi(j),2}] = 1\ \Big) && (10)\end{aligned}$$

Equivalency (8) holds because by definition of Z', for all i and k, $Z[i, \phi^{-1}(k)] = Z'[i, k]$, so $Z[i, k] = Z'[i, \phi(k)]$. Equivalency (9) holds because $(\psi(1), \psi(3)) = (1, 3)$ or $(3, 1)$, so $(Z'[i, \phi(t_{j,1})] \neq Z'[i, \phi(t_{j,3})]) \Leftrightarrow (Z'[i, \phi(t_{j,\psi(1)})] \neq Z'[i, \phi(t_{j,\psi(3)})])$. Finally, Equivalency (10) holds because by definition of Σ', we have that $(\phi(t_{j,\psi(1)}), \phi(t_{j,2}), \phi(t_{j,\psi(3)})) = (t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3})$. ◁

We show that (Z is a valid solution for Σ) $\Leftrightarrow$ (Z' is a valid solution for Σ'). By definition, (Z is a valid solution for Σ) is equivalent to the following property :

Property 1: $f = Z[1]$, $l = Z[S]$, and for any $i \in [\![1, S-1]\!]$, there exists $j \in [\![1, T]\!]$ such that, parsing t_j as $(t_{j,1}, t_{j,2}, t_{j,3})$:
- for any $k \in [\![1, N]\!]$, $(k \in \{t_{j,1}, t_{j,2}, t_{j,3}\} \Leftrightarrow Z[i,k] \neq Z[i+1,k])$, and
- $Z[i, t_{j,1}] \neq Z[i, t_{j,3}]$ and $Z[i, t_{j,2}] = 1$.

By Claim 11, 12 and 13, Property 1 is equivalent to the following property:

Property 2: $f' = Z'[1]$, $l' = Z'[S]$, and for any $i \in [\![1, S-1]\!]$, there exists $j \in [\![1, T]\!]$ such that, parsing $t'_{\chi(j)}$ as $(t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3})$:
- for any $k \in [\![1, N]\!]$, $(k \in \{t'_{\chi(j),1}, t'_{\chi(j),2}, t'_{\chi(j),3}\} \Leftrightarrow Z'[i,k] \neq Z'[i+1,k])$, and
- $Z'[i, t'_{\chi(j),1}] \neq Z'[i, t'_{\chi(j),3}]$ and $Z'[i, t'_{\chi(j),2}] = 1$.

χ is a bijection, so $\{\chi(j)\}_{j \in [\![1,T]\!]} = [\![1, T]\!]$, which implies that Property 2 is equivalent to the following property:

Property 3: $f' = Z'[1]$, $l' = Z'[S]$, and for any $i \in [\![1, S-1]\!]$, there exists $j \in [\![1, T]\!]$ such that, parsing t'_j as $(t'_{j,1}, t'_{j,2}, t'_{j,3})$:
- for any $k \in [\![1, N]\!]$, $(k \in \{t'_{j,1}, t'_{j,2}, t'_{j,3}\} \Leftrightarrow Z'[i,k] \neq Z'[i+1,k])$, and
- $Z'[i, t'_{j,1}] \neq Z'[i, t'_{j,3}]$ and $Z'[i, t'_{j,2}] = 1$.

Finally, by definition, Property 3 is equivalent to (Z' is a valid solution for Σ'), which concludes the proof. ◀

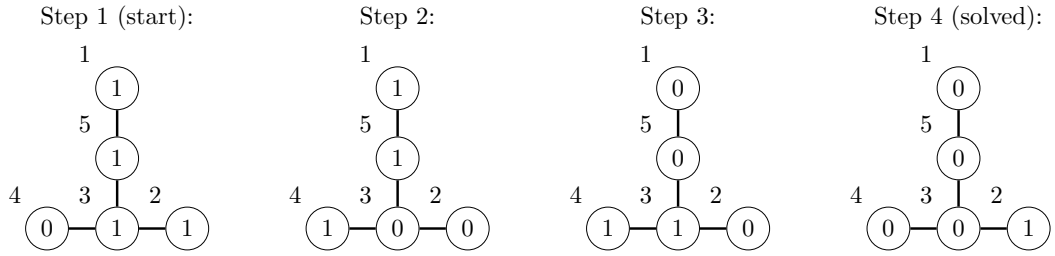

Figure 5 The graph representation of the steps of the solution for a peg solitaire which is isomorphic to the tee peg solitaire, by the isomorphism given in Example 14.

▶ **Example 14.** Let $\xi = (\phi, \chi, \psi)$ be a $(5, 2, 4)$-peg solitaire isomorphism defined by:
- For any $x \in [\![1, N]\!]$, $\phi(x) = (x \mod 5) + 1$.
- $\chi(1) = 2$ and $\chi(2) = 1$.
- $\psi(1) = 3$ and $\psi(3) = 1$.

The $(5, 2, 4)$-peg solitaire instance Σ' isomorphic to the tee peg solitaire instance (given in Example 5) by ξ is defined by:
- $t' = ((1, 5, 3), (4, 3, 2))$.
- $f' = (1, 1, 1, 0, 1)$ and $l' = (0, 1, 0, 0, 0)$.

Using Theorem 10 on the valid solution for the tee peg solitaire (given in Example 7), we obtain the following valid solution Z' for Σ':

$$Z' = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

We give the graph representation of this solution in Figure 5.

5 Protocol

In this section we present our proof protocol for the peg solitaire. This protocol has a sigma structure, it therefore consists of three interractions between a prover, knowing a solution Z for a (N,T,S)-peg solitaire instance Σ, and a verifier, knowing only Σ. We first give a high-level description of the protocol :

- **The prover** chooses an (N,T,S)-isomorphism ξ and computes the peg solitaire instance Σ', isomorphic to Σ by the isomorphism ξ, together with a valid solution Z' of Σ' (using the method given in Theorem 10). The prover commits each part of Σ' and Z', and sends the commitments to the verifier.
- **The verifier** chooses $\sigma \in [\![1,S]\!]$ at random, and sends it to the prover.
- **The prover** computes the response in terms of σ :
  - **If $\sigma = S$,** then the prover reveals the isomorphism ξ and the peg solitaire instance Σ' to the verifier. The verifier checks the validity of the commitments of Σ', and checks that Σ' is isomorphic to Σ by the isomorphism ξ.
  - **Else,** the prover reveals the integer $\gamma \in [\![1,T]\!]$, which is the index of the triplet t'_γ in Σ' that contains the indexes of the vertices that change their values between step σ and step $(\sigma+1)$ of the solution. Such an integer always exists by definition of a valid solution. The prover reveals t'_γ and reveals the lines σ and $(\sigma+1)$ of the solution Z' (*i.e.* the vectors that correspond to the state of the peg solitaire before and after the σ^{th} move) to the verifier. The verifier checks the validity of the commitments of t'_γ and of the two revealed lines of Z', and checks that the σ^{th} move is legal according to the triplet t'_γ.

  If all the checks are valid, then the verifier accepts the transcript, else it rejects it.

A classical variant of peg solitaire consists in not imposing a final position, but only the number of pegs that must be removed from the board. We note that our protocol can be adapted to this (less constraining) version of the game simply by not revealing the vector of the final position in the case $\sigma = S$.

The formal definition of the protocol is given in Protocol 1. In Corollary 16, we show that this protocol is complete, valid, and zero-knowledge under the hypothesis that the commitment scheme has both perfect binding and hiding properties.

▶ **Protocol 1.** *Let $\mathcal{P}$ be a prover knowing a solution Z for a (N,T,S)-peg solitaire instance $\Sigma = (t,f,l)$, and $\mathcal{V}$ be a verifier knowing Σ. Our protocol uses a commitment scheme $C = (\mathsf{Gen}, \mathsf{Commit})$ and a security parameter λ. Our protocol is a sigma protocol having the following successive interactions, repeated λ times:*

- **The prover** *$\mathcal{P}$ chooses an isomorphism $\xi = (\phi,\chi,\psi)$ at random, and computes the (N,T,S)-peg solitaire instance $\Sigma' = (t',f',l')$, isomorphic to Σ by ξ. The prover $\mathcal{P}$ computes a valid solution Z' for Σ' by using the method given in Theorem 10 on Σ, Z and ξ. For all $i \in [\![1,T]\!]$, and all $j \in [\![1,S]\!]$:*
  - *$\mathcal{P}$ generates $\mathsf{skt}_i \leftarrow \mathsf{Gen}(1^\lambda)$ and computes the commitment $\widehat{t_i} = \mathsf{Commit}(\mathsf{skt}_i, t'_i)$.*
  - *$\mathcal{P}$ generates $\mathsf{skz}_j \leftarrow \mathsf{Gen}(1^\lambda)$ and computes $\widehat{Z}_j = \mathsf{Commit}(\mathsf{skz}_j, Z'[j])$.*

  Finally, $\mathcal{P}$ sends $(\widehat{t_i})_{1\le i\le T}$ and $(\widehat{Z}_j)_{1\le j\le S}$ to the verifier.
- **The verifier** *$\mathcal{V}$ chooses $\sigma \in [\![1,S]\!]$ at random, and sends it to the prover.*
- **The prover** *$\mathcal{P}$ computes the response in terms of σ :*
  - **If $\sigma = S$,** *then $\mathcal{P}$ sends ξ, $(\mathsf{skt}_i, t'_i)_{1\le i\le T}$, $(\mathsf{skz}_1, Z'[1])$, and $(\mathsf{skz}_S, Z'[S])$ to the verifier.*
  - **Else,** *the prover $\mathcal{P}$ chooses the integer $\gamma \in [\![1,T]\!]$, which is the index of the triplet t'_γ containing the indexes of the vertices that change their values between step σ and step*

$(\sigma+1)$ *of the solution (such an integer always exists by definition of a valid solution).* $\mathcal{P}$ *then sends* $(\mathsf{skt}_\gamma, t'_\gamma)$, $(\mathsf{skz}_\sigma, Z'[\sigma])$ *and* $(\mathsf{skz}_{\sigma+1}, Z'[\sigma+1])$ *to the verifier.*

- **The verifier***'s verification depends on the value of* σ*:*
  - **If** $\sigma = S$**,** *then* $\mathcal{V}$ *verifies the following commitments:*
    - *For* $i \in [\![1, T]\!]$, $\mathcal{V}$ *checks that* $\widehat{t}_i = \mathsf{Commit}(\mathsf{skt}_i, t'_i)$,
    - $\mathcal{V}$ *checks that* $\widehat{Z}_1 = \mathsf{Commit}(\mathsf{skz}_1, Z'[1])$.
    - $\mathcal{V}$ *checks that* $\widehat{Z}_S = \mathsf{Commit}(\mathsf{skz}_S, Z'[S])$.

    $\mathcal{V}$ *then checks that* $((t'_i)_{1\leq i\leq T}, Z'[1], Z'[S])$ *is isomorphic to* Σ *by* ξ.
  - **Else,** $\mathcal{V}$ *verifies the following commitments:*
    - $\mathcal{V}$ *checks that* $\widehat{t}_\gamma = \mathsf{Commit}(\mathsf{skt}_\gamma, t'_\gamma)$.
    - $\mathcal{V}$ *checks that* $\widehat{Z}_\sigma = \mathsf{Commit}(\mathsf{skz}_\sigma, Z'[\sigma])$.
    - $\mathcal{V}$ *checks that* $\widehat{Z}_{\sigma+1} = \mathsf{Commit}(\mathsf{skz}_{\sigma+1}, (Z'[\sigma+1])$.

    $\mathcal{V}$ *then checks that:*
    - *for any* $k \in [\![1, N]\!]$, $(k \in \{t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}\} \Leftrightarrow Z'[\sigma, k] \neq Z'[\sigma+1, k])$, *and*
    - $Z'[\sigma, t'_{\gamma,1}] \neq Z'[\sigma, t'_{\gamma,3}]$ *and* $Z'[\sigma, t'_{\gamma,2}] = 1$.

  If all the checks are valid, then $\mathcal{V}$ *accepts the transcript, else* $\mathcal{V}$ *rejects it.*

▶ **Theorem 15.** *If the sigma protocol given in Protocol 1 is instantiated with a commitment scheme having the hiding and the perfect binding properties, then it is complete,* S*-special-sound, and (computational) zero-knowledge, where* S *is the number of steps in the solution of the peg solitaire instance.*

As mentioned in Section 2, if a sigma protocol repeated λ times is n-special-sound, then it is a valid proof of knowledge whose the knowledge error is $((n-1)/n)^\lambda$, which implies the following corollary.

▶ **Corollary 16.** *If Protocol 1 is instantiated with a commitment scheme having the hiding and the perfect binding properties, then it is a complete, valid, and (computational) zero-knowledge proof of knowledge whose the knowledge error is* $((S-1)/S)^\lambda$*, where* S *is the number of steps in the solution of the peg solitaire instance and* λ *is the number of rounds of the protocol.*

In what follows, we prove Theorem 15.

Proof. First, let us present a sketch of the proof.

Completeness: The prover knows a valid solution for Σ, so it is able to compute a peg solitaire instance isomorphic to Σ and its valid solution using the method described in Theorem 10. Since the solution of the isomorphic instance is valid, the prover can always answer the verifier correctly.

S-Special-soundness: Assume that the prover is able to respond correctly whatever the challenge it receives for the same commitment. Since the commitment scheme has the perfect binding property, the prover opens its commitments in a consistent way (*i.e.* the same commitment is not valid for two different values). We have that:

- The response to the challenge S reveals an isomorphism ξ and a peg solitaire instance Σ' that is isomorphic to Σ by ξ.
- The response to each challenge $\sigma < S$ reveals one move of the valid solution Z' for Σ'.

By putting all the moves of Z' together, we can extract the whole solution Z'. Using Theorem 10, we extract a valid solution Z for Σ from Z' and the isomorphism ξ.

Zero-knowledge: First, assume that the commitments perfectly hide the committed values.

- If the verifier chooses the challenge S, then it receives an isomorphism ξ randomly chosen, and the image of Σ by ξ.

  In this case the verifier can simulate the protocol by choosing a random isomorphism.

- If the challenger chooses a challenge $\sigma < S$, then it receives a triplet $t'_\gamma = (x, y, z)$ of Σ' and two successive vectors of the solution Z'. The index of the triplet depends on the unknown random permutation χ, and seems to be randomly chosen in $[\![1, T]\!]$ from the verifier point of view. Since the indexes of the nodes are shuffled by the unknown random permutation ϕ, the triplet (x, y, z) contains three different integers in $[\![1, N]\!]$ that seem randomly chosen from the verifier point of view. Let F be the number of 1 in the vector f of Σ; since at each step of the solution only one peg is removed from the board, the number of 1 in the first revealed vector of the solution is $F + \sigma - 1$. Note that this property does not depend on the valid solution, and does not need to be hidden from the verifier. Except for the bits indexed by x, y and z, the 1 seem to be randomly distributed in the vector from the verifier point of view, because the indexes of the nodes are shuffled by the unknown random permutation ϕ. According to the verification requirements, the three bits indexed by x, y and z have a specific structure in the two vectors: they must match one of the two positions (●●○) or (○●●) in the first vector, and (○ ○ ●) or (● ○ ○) respectively in the second one. The chosen position depends on the unknown random permutation ψ and seems to be randomly chosen from the verifier point of view. The other bits are the same in the two vectors. In this case the verifier can simulate the protocol by choosing the index of a triplet in $[\![1, T]\!]$ at random, choosing three random indexes in $[\![1, N]\!]$ at random, putting the bits of these three indexes as in one of the two legal positions (chosen at random) in the two vectors of the solution, and choosing the joint positions of the other 1 in the two vectors at random.

In both case, the verifier simulates the unrevealed commitments by committing a random bit-string, so the real protocol is perfectly simulated. Now, consider that the commitments do not perfectly hide the committed values. In this case, the zero-knowledge property depends on the ability of the verifier to distinguish which value is hidden in a commitment. Zero-knowledge is thus ensured by the hiding property of the commitment scheme.

We now give the technical details for the proof of each propertie.

▷ Claim 17. The protocol is complete.

Proof. The honest prover knows the (N, T, S)-isomorphism ξ and the committed (N, T, S)-peg solitaire instance Σ', which is isomorphic to Σ by ξ. It also knows the committed valid solution Z' for Σ'. We recall that according to Theorem 10, a honest prover knowing Z, Σ and ξ is able to efficiently generate a valid solution Z' for Σ', because it is isomorphic to Σ by ξ.

If the challenge is S, then it is clear that the verifier accepts the proof of an honest prover revealing its isomorphism.

By definition of a valid solution (Definition 6), if $Z' = (t', f', l')$ is a solution for Σ', then for any $\sigma \in [\![1, S-1]\!]$, there exists $\gamma \in [\![1, T]\!]$ such that, parsing t'_γ as $(t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3})$:

- For any $k \in [\![1, N]\!]$, $(k \in \{t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}\} \Leftrightarrow Z'[\sigma, k] \neq Z'[\sigma + 1, k])$.
- $Z'[\sigma, t'_{\gamma,1}] \neq Z'[\sigma, t'_{\gamma,3}]$ and $Z'[\sigma, t'_{\gamma,2}] = 1$.

Thus, for any challenge $\sigma \in [\![1, S-1]\!]$, by choosing the index γ, the honest prover convinces the verifier, which concludes the proof of completeness. ◁

▷ Claim 18. If the commitment scheme C has the perfect binding property, then the protocol is S-special-sound.

Proof. Assume that a prover gives an accepted response for any challenge $\sigma \in [\![1, S]\!]$ on the same commitment. Since the commitment scheme is assumed to have the binding property, the same commitment cannot be opened in several different ways in the S responses of the prover. We show how to build a polynomial time knowledge extractor that computes a valid solution Z for Σ from these transcripts.

Since all responses are accepted, all checks required during the protocol are valid, which implies that:

- $\Sigma' = ((t'_i)_{1\leq i\leq T}, Z'[1], Z'[S])$ is isomorphic to Σ by $\xi = (\phi, \chi, \psi)$.
- for each challenge $\sigma \in [\![1, S-1]\!]$, the prover reveals an index $\gamma \in [\![1, T]\!]$ such that, parsing t'_γ as $(t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3})$:
  - For any $k \in [\![1, N]\!]$, $(k \in \{t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}\} \Leftrightarrow Z'[\sigma, k] \neq Z'[\sigma+1, k])$.
  - $Z'[\sigma, t'_{\gamma,1}] \neq Z'[\sigma, t'_{\gamma,3}]$ and $Z'[\sigma, t'_{\gamma,2}] = 1$.

By definition of a valid solution (Definition 6), we deduce that Z' is a valid solution for Σ'.

Let $Z = (Z[i,j])_{1\leq i\leq S; 1\leq j\leq N}$ be a solution for Σ such that, for any $i \in [\![1, S]\!]$ and $j \in [\![1, N]\!]$, $Z[i,j] = Z'[i, \phi(j)]$. Our knowledge extractor computes and returns this solution Z. We have that for any $i \in [\![1, S]\!]$ and $j \in [\![1, N]\!]$, $Z'[i,j] = Z[i, \phi^{-1}(j)]$, we deduce that Z is a valid solution for Σ according to Theorem 10, which concludes the proof. ◁

▷ Claim 19. If the commitment scheme C has the hiding property, then the protocol is zero-knowledge.

Proof. In what follows, by abuse of language, we use the expression *"pick in X at random"* for *"pick from the uniform distribution on X"*. Let $\mathcal{V}^*$ be a p.p.t verifier, Σ be a (N, T, S)-peg solitaire instance, and Z be a valid solution for Σ. Let str be a bit-string and p be a polynomial. We define the simulator $\mathsf{Sim}(\Sigma)$ for $\mathcal{V}^*$ as follows, where $\Sigma = (t, f, l)$ is a (N, T, S)-peg solitaire instance.

The simulator. Sim picks $\sigma \in [\![1, S]\!]$ at random. We distinguish the two following cases:

- If $\sigma = S$, Sim chooses an isomorphism $\xi = (\phi, \chi, \psi)$ at random, and computes the (N, T, S)-peg solitaire instance $\Sigma' = (t', f', l')$ isomorphic to Σ by ξ. Sim then generates the following commitments:
  - For all $i \in [\![1, T]\!]$, Sim generates $\mathsf{skt}_i \leftarrow \mathsf{Gen}(1^\lambda)$ and computes the commitment $\widehat{t}_i = \mathsf{Commit}(\mathsf{skt}_i, t'_i)$.
  - Sim generates $\mathsf{skz}_1 \leftarrow \mathsf{Gen}(1^\lambda)$ and computes $\widehat{Z}_1 = \mathsf{Commit}(\mathsf{skz}_1, f'_1)$.
  - For all $j \in [\![2, S-1]\!]$, Sim generates $\mathsf{skz}_j \leftarrow \mathsf{Gen}(1^\lambda)$ and computes $\widehat{Z}_j = \mathsf{Commit}(\mathsf{skz}_j, \mathsf{str})$.
  - Sim generates $\mathsf{skz}_S \leftarrow \mathsf{Gen}(1^\lambda)$ and computes $\widehat{Z}_S = \mathsf{Commit}(\mathsf{skz}_S, l'_S)$.

  Finally, Sim sets the prover commitment as $\mathsf{com} = ((\widehat{t}_i)_{1\leq i\leq T}, (\widehat{Z}_j)_{1\leq j\leq S})$, the verifier challenge as σ, and the prover response as $\mathsf{res} = (\xi, (\mathsf{skt}_i, t'_i)_{1\leq i\leq T}, (\mathsf{skz}_1, f'), (\mathsf{skz}_S, l'))$.
- Else, Sim picks $\gamma \in [\![1, T]\!]$ and picks a triplet $t'_\gamma = (t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}) \in [\![1, N]\!]^3$ at random that verifies $t'_{\gamma,1} \neq t'_{\gamma,2}$, $t'_{\gamma,2} \neq t'_{\gamma,3}$ and $t'_{\gamma,1} \neq t'_{\gamma,3}$. Let F be the number of 1 in the vector f of Σ. Sim picks a bit b at random, then Sim picks a vector $Z'[\sigma]$ of N bits at random such that $Z'[\sigma]$ verifies the following properties:
  - $Z'[\sigma]$ contains exactly $F + \sigma - 1$ times the bit 1, and
  - $Z'[\sigma, t'_{\gamma,1}] = b$, $Z'[\sigma, t'_{\gamma,2}] = 1$, and $Z'[\sigma, t'_{\gamma,3}] = 1 - b$.

  Sim sets a vector $Z'[\sigma+1]$ of N bits such that for all $k \in [\![1, N]\!]$, $(k \in \{t'_{\gamma,1}, t'_{\gamma,2}, t'_{\gamma,3}\} \Leftrightarrow Z'[\sigma, k] \neq Z'[\sigma+1, k])$. Sim then generates the following commitments:
  - For all $i \in [\![1, T]\!] \backslash \{\gamma\}$, Sim generates $\mathsf{skt}_i \leftarrow \mathsf{Gen}(1^\lambda)$ and computes the commitment $\widehat{t}_i = \mathsf{Commit}(\mathsf{skt}_i, \mathsf{str})$.

- Sim generates $\mathsf{skt}_\gamma \leftarrow \mathsf{Gen}(1^\lambda)$ and computes the commitment $\widehat{t}_\gamma = \mathsf{Commit}(\mathsf{skt}_\gamma, t'_\gamma)$.
- For all $j \in [\![1, S]\!] \backslash \{\sigma, \sigma+1\}$, Sim generates $\mathsf{skz}_j \leftarrow \mathsf{Gen}(1^\lambda)$ and computes $\widehat{Z}_j = \mathsf{Commit}(\mathsf{skz}_j, \mathsf{str})$.
- Sim generates $\mathsf{skz}_\sigma \leftarrow \mathsf{Gen}(1^\lambda)$ and computes $\widehat{Z}_\sigma = \mathsf{Commit}(\mathsf{skz}_\sigma, Z'[\sigma])$.
- Sim generates $\mathsf{skz}_{\sigma+1} \leftarrow \mathsf{Gen}(1^\lambda)$ and computes $\widehat{Z}_{\sigma+1} = \mathsf{Commit}(\mathsf{skz}_{\sigma+1}, Z'[\sigma+1])$.

Finally, Sim sets the prover commitment as $\mathsf{com} = ((\widehat{t}_i)_{1\leq i\leq T}, (\widehat{Z}_j)_{1\leq j\leq S})$, the verifier challenge as σ, and the prover response as $\mathsf{res}((\mathsf{skt}_\gamma, t'_\gamma), (\mathsf{skz}_\sigma, Z'[\sigma]), (\mathsf{skz}_{\sigma+1}, Z'[\sigma+1]))$.

Finally, Sim runs a proof protocol with $\mathcal{V}^*(\Sigma)$, sends com, receives a challenge σ', and aborts the protocol. If $\sigma' = \sigma$, then Sim returns $(\mathsf{com}, \sigma, \mathsf{res})$, else, the simulator starts all over again with a new random σ. After $p(\lambda)$ unsuccessful tries, Sim returns the failure symbol $\perp$.

We show that no p.p.t distinguisher $\mathcal{D}$ is able to distinguish between real transcripts from the protocol and transcripts simulated by Sim. We use a sequence of game between $\mathcal{D}$ and a p.p.t challenger $\mathcal{C}$ as described by Shoup in [17]. In what follows, $\epsilon(\lambda)$ denotes the probability of the best algorithm attacking the hiding property of the commitment scheme (which is assumed to be negligible).

Game $\mathbf{G_0}$: In this game, the challenger $\mathcal{C}$ runs the real protocol $\mathsf{tr} \leftarrow \langle \mathcal{P}(Z), \mathcal{V}^*(\Sigma)\rangle$, and runs $b \leftarrow \mathcal{D}(\mathsf{tr})$.

Game $\mathbf{G_1}$: We define the following algorithm:

- $\mathcal{P}_1(Z)$ is the same algorithm as $\mathcal{P}$ except that it starts by picking a challenge $\sigma \in [\![1, S]\!]$ at random, then it aborts if the verifier chooses a challenge σ' such that $\sigma' = \sigma$.

In this game, the challenger $\mathcal{C}$ runs the protocol $\mathsf{tr} \leftarrow \langle \mathcal{P}_1(Z), \mathcal{V}^*(\Sigma)\rangle$ until the prover does not abort. If $\mathcal{P}_1$ aborts $p(\lambda)$ times successively, then $\mathcal{C}$ sets $\mathsf{tr} = \perp$. Since $\mathcal{P}_1$ aborts with probability $(S-1)/S$, the challenger $\mathcal{C}$ sets $\mathsf{tr} = \perp$ with probability $((S-1)/S)^{p(\lambda)}$. We have that:

$$|\mathsf{Pr}[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G_0}] - \mathsf{Pr}[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G_1}]| \leq \mathsf{Pr}[\mathcal{C} \text{ sets } \mathsf{tr} = \perp \text{ in } \mathbf{G_1}] = \left(\frac{S-1}{S}\right)^{p(\lambda)}.$$

We note that from this game, **the prover knows in advance (*i.e.*, before generating the commitments) which commitments will be revealed and which will not**. We also recall that the prover generates $S+T$ commitments during the protocol.

Game $\mathbf{G_{1,i}}$ (for $\mathbf{0 \leq i \leq S+T}$): We define the following algorithm:

- $\mathcal{P}_{1,i}(Z)$ is the same algorithm as $\mathcal{P}_1$ except that for the first i commitments that will not be revealed to the verifier in the response phase, the prover $\mathcal{P}_{1,i}$ commits the value str instead of the value that is committed in the real protocol.

This game is the same as $\mathbf{G_1}$ except that it uses $\mathcal{P}_{1,i}$ instead of $\mathcal{P}_1$. Clearly, $\mathbf{G_1} = \mathbf{G_{1,0}}$. We claim that, for all $i \in [\![1, S+T-1]\!]$:

$$|\mathsf{Pr}[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G_{1,i}}] - \mathsf{Pr}[\mathcal{D} \text{ returns } 1 \text{ in } \mathbf{G_{1,i+1}}]| \leq \epsilon(\lambda).$$

We prove this claim by reduction. We use $\mathcal{D}$ to build a p.p.t algorithm $\mathcal{A}$ that plays the experiment $\mathbf{Exp}^{\mathsf{Hiding}}_{C,\mathcal{A},b'}(\lambda)$. The algorithm $\mathcal{A}$ plays the role of $\mathcal{C}$ and simulates $\mathsf{tr} \leftarrow \langle \mathcal{P}_{1,i}(Z), \mathcal{V}^*(\Sigma)\rangle$ exactly as in $\mathbf{G_{1,i}}$, except that $\mathcal{A}$ sets m_0 as the value that is committed in the $(i+1)^{\text{th}}$ unrevealed commitment produced by $\mathcal{P}_{1,i}$, sets $m_1 = \mathsf{str}$, sends (m_0, m_1) to the experiment $\mathbf{Exp}^{\mathsf{Hiding}}_{C,\mathcal{A},b'}(\lambda)$, receives the commitment $c_{b'}$ of $m_{b'}$, and replaces the $(i+1)^{\text{th}}$

unrevealed commitment of $\mathcal{P}_{1,i}$ by $c_{b'}$ in tr. (Obviously, $\mathcal{C}$ replaces this commitment in the transcript of the protocol instance that did not abort). Finally, $\mathcal{A}$ runs $b \leftarrow \mathcal{D}(\mathsf{tr})$ and returns b to the experiment. We have that:

$$|\mathsf{Pr}[\mathcal{D} \text{ returns 1 in } \mathbf{G_{1,i}}] - \mathsf{Pr}[\mathcal{D} \text{ returns 1 in } \mathbf{G_{1,i+1}}]| = \left|\mathsf{Pr}\left[0 \leftarrow \mathbf{Exp}^{\mathsf{Hiding}}_{C,\mathcal{A},0}(\lambda)\right] - \mathsf{Pr}\left[1 \leftarrow \mathbf{Exp}^{\mathsf{Hiding}}_{C,\mathcal{A},1}(\lambda)\right]\right| \leq \epsilon(\lambda),$$

which concludes the proof of the claim. We note that from the game $\mathbf{G_{1,T+S}}$, **all the commitments that are not revealed to the verifier commit the value** str **as in our simulator.**

Game $\mathbf{G_2}$: In this game, $\mathcal{C}$ runs $\mathsf{tr} \leftarrow \mathcal{S}(\Sigma)$ and $b \leftarrow \mathcal{D}(\mathsf{tr})$. We have that $\mathbf{G_{1,T+S}} = \mathbf{G_2}$ (the justification of this equivalence has already been given in the sketch of the proof). Finally, using each of these game hops, we deduce that:

$$\begin{aligned}&|\mathsf{Pr}[\mathsf{tr} \leftarrow \langle \mathcal{P}(Z), \mathcal{V}^*(\Sigma)\rangle; b \leftarrow \mathcal{D}(\mathsf{tr}) : b = 1] - \mathsf{Pr}[\mathsf{tr} \leftarrow \mathsf{Sim}(\Sigma); b \leftarrow \mathcal{D}(\mathsf{tr}) : b = 1]| \\ &= |\mathsf{Pr}[\mathcal{D} \text{ returns 1 in } \mathbf{G_0}] - \mathsf{Pr}[\mathcal{D} \text{ returns 1 in } \mathbf{G_2}]| \leq (S+T)\epsilon(\lambda) + \left(\frac{S-1}{S}\right)^{p(\lambda)}.\end{aligned}$$

Since ϵ is negligible and p is a polynomial, this value is negligible, which concludes the proof. ◁

◀

6 Conclusion

In this paper, we have given a tailor-made zero-knowledge proof of knowledge for the peg solitaire, which does not use any reduction to another NP-complete problem. Our proof can be used physically by using envelopes instead of commitments. We also define the notion of peg solitaire isomorphism, which is of independent interest. In the future, it would be interesting to reduce the soundness error of our protocol, either by reducing the number of challenges in the cryptographic version of the protocol, or by using techniques that are specific to the physical proof protocols, such as the one introduced in [16]. Another possible future work would be to extend our protocol to the many variants of the peg solitaire, and more generally, to other single-player board games.

References

1 David Avis and Antoine Deza. On the solitaire cone and its relationship to multi-commodity flows. *Mathematical Programming*, 90:27–57, March 2001. doi:10.1007/PL00011419.

2 Robert A. Beeler and D. Paul Hoilman. Peg solitaire on graphs. *Discrete Mathematics*, 311(20):2198–2202, 2011. doi:10.1016/j.disc.2011.07.006.

3 Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 390–420, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

4 Xavier Bultel, Jannik Dreier, Jean-Guillaume Dumas, and Pascal Lafourcade. Physical Zero-Knowledge Proofs for Akari, Takuzu, Kakuro and KenKen. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms (FUN 2016)*, volume 49 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:20, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FUN.2016.8.

5 Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, pages 174–187, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

6 Ivan Damgård. *Commitment Schemes and Zero-Knowledge Protocols*, pages 63–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. doi:10.1007/3-540-48969-X_3.

7 Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, July 1991. doi:10.1145/116825.116852.

8 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, New York, NY, USA, 1985. Association for Computing Machinery. doi:10.1145/22145.22178.

9 Ronen Gradwohl, Moni Naor, Benny Pinkas, and Guy N. Rothblum. Cryptographic and physical zero-knowledge proof systems for solutions of sudoku puzzles. In *Proceedings of the 4th International Conference on Fun with Algorithms*, FUN'07, pages 166–182, Berlin, Heidelberg, 2007. Springer-Verlag.

10 Luciano Gualà, Stefano Leucci, Emanuele Natale, and Roberto Tauraso. Large Peg-Army Maneuvers. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms (FUN 2016)*, volume 49 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FUN.2016.18.

11 Christopher Jefferson, Angela Miguel, Ian Miguel, and S. Armagan Tarim. Modelling and solving english peg solitaire. *Computers & Operations Research*, 33(10):2935–2959, 2006. Part Special Issue: Constraint Programming. doi:10.1016/j.cor.2005.01.018.

12 Masashi Kiyomi and Tomomi Matsui. Integer programming based algorithms for peg solitaire problems. In Tony Marsland and Ian Frank, editors, *Computers and Games*, pages 229–240, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

13 Pascal Lafourcade, Daiki Miyahara, Takaaki Mizuki, Léo Robert, Tatsuya Sasaki, and Hideaki Sone. How to construct physical zero-knowledge proofs for puzzles with a "single loop" condition. *Theoretical Computer Science*, 888:41–55, 2021. doi:10.1016/j.tcs.2021.07.019.

14 Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.

15 Bala Ravikumar. Peg-solitaire, string rewriting systems and finite automata. *Theor. Comput. Sci.*, 321(2–3):383–394, August 2004. doi:10.1016/j.tcs.2004.05.005.

16 Tatsuya Sasaki, Daiki Miyahara, Takaaki Mizuki, and Hideaki Sone. Efficient card-based zero-knowledge proof for sudoku. *Theoretical Computer Science*, 839:135–142, 2020. doi:10.1016/j.tcs.2020.05.036.

17 Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. URL: https://ia.cr/2004/332.

18 Ryuhei Uehara and Shigeki Iwata. Generalized hi-q is np-complete. *Transactions of the IEICE*, 1990.

19 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*, pages 74–91, Cham, 2013. Springer International Publishing.