Physical Zero-knowledge Proofs for Flow Free, Hamiltonian Cycles, and Many-to-many k-disjoint Covering Paths

Eammon Hart and Joshua A. McGinnis

February 10, 2022

Abstract

In this paper we describe protocols which use a standard deck of cards to provide a perfectly sound zero-knowledge proof for Hamiltonian cycles and Flow Free puzzles. The latter can easily be extended to provide a protocol for a zero-knowledge proof of many-to-many k-disjoint path coverings.

Keywords — Flow Free, Hamiltonian cycles, disjoint covering paths, ZKP, card-based cryptography, Numberlink, graph, puzzle

1 Introduction

Hamiltonian cycles have been a go to example for introductions to zero-knowledge proofs for decades [2], [3], but as far as we can tell all existing proofs are probabilistic. Unlike those methods in this paper we use a deck of cards to provide a perfectly sound zero-knowledge proof for the possession of a Hamiltonian cycle, i.e. the probability of deception is 0.

Flow Free is a popular logic puzzle app game which has over 100,000,000 downloads in the Google play story [4] and whose spin off variants of Flow Free Hexes [6] and Flow Free Bridges [5] have millions of downloads as well. They are variants of the Numberlink puzzle which dates back to at least 1897 when it was published by Sam Loyd. A similar game was published by Nikoli, the Japanese publisher famous for Sudoku. Flow Free involves an $n \times m$ grid with labeled cells, such that one must connect given pairs of cells with paths that collectively cover all of the cells. The Nikoli variant places the additional requirement of minimizing the turns in the paths connecting the given pairs of cells, but drops the requirement that every cell must be filled. This means that solution paths between connected pair of cells may need to be non-simple, which adds some additional complexity to the problem of finding a zero-knowledge proof. Flow Free also relates deeply to research on a generalization of the problem of finding a Hamiltonian path with given starting points called the disjoint covering paths problem (studied by [17], [10], [23], [16], and [13] among many others), because the solution to each game can easily be mapped to a disjoint covering path for the underlying adjacency graph of the game. Numberlink corresponds to the question of
vertex-disjoint paths that do not necessarily cover every vertex. Past study of the
game has primarily been from the computer science direction, with solving Flow Free
and Zig Zag Numberlink being shown to be NP Complete [1], algorithmic explorations
enumerating all solutions to a board, identifying games with a unique solution in [22],
and efficiently solving puzzles using Monte Carlo methods [9]. [14] has also recently
used it in assessing neurological degeneration. More directly related to our paper
[19] provides a physical zero-knowledge proof of solutions to games of Numberlink
and more broadly, of knowledge of k vertex-disjoint paths, but they do not provide
a zero-knowledge proof for solutions to games Flow Free or the setting of knowledge
of k vertex-disjoint paths which cover the vertices (they explicitly identify this as a
potential avenue for future research). In this paper we provided several extensions
to the methodology in [19] that allow us to construct perfectly sound zero-knowledge
proofs for Hamiltonian cycles, games of Flow Free, and general many-to-many k-disjoint
path covers. The main obstacle that we overcome is in dealing with non-simple paths,
i.e. a vertex in a path is adjacent to another vertex in the same path but the vertices
are not path adjacent; see Figure 1. It should also be noted that this paper sits in a
young, but growing literature of perfectly sound physical zero-knowledge proofs that
also includes Zig-zag Numberlink [19], Ripple Effect [20], Sudoku [21] (this differs from
previous non-certain zero-knowledge proofs for Sudoku provided by [7]), Kakuro [11],
Takuzo and Jousan [12], Nurikabe and Hitori [18], among others.

2 Hamiltonian Cycles

Methods for providing zero-knowledge proofs for Hamiltonian cycles dates back at
least to [2] and have been elaborated in other settings such as in the non-interactive
case in [3], but as far we can tell all zero-knowledge proofs for Hamiltonicity that
currently exist in the literature are not perfectly sound. For example Blum’s protocol,
which seems to be the canonical one, gives probability $2^{-n}$ that the prover does not
actually know a Hamiltonian cycle, where \( n \) is the number of checks performed. Our protocol can therefore be seen as an improvement on the previous ones by providing a zero-knowledge proof with perfect soundness.

2.1 The Protocol

In what follows, cards are used for indexing matrices and encoding information for edges in the graph. The encoding is done via two suits ♥ (hearts) and ♠ (spades). Indexing can be done by using the numerical value of the cards (for our purposes, since the number of indexing cards needs to be large, we assume our deck of cards has relatively large values such as the rare 30 of clubs.)

We use an encoding system similar to that found in [19] to encode numbers. Define \( E_k(i) \) to be a sequence of \( k \) encoding (marking) cards in a row, with the \( i \)th card being a heart and all other cards being spades. If \( i = 0 \), then let the entire sequence of cards be spades. For examples \( E_3(2) = ♥♥♠ \), while \( E_3(3) = ♥♠♥♠ \). For the purposes of checking Hamiltonicity we only need to use \( E_1(1) = ♥, E_1(0) = ♠, \) and \( E_n(k) \) for all \( 0 \leq k \leq n \) where \( n \) is the number of vertices in the graph. Cards are often laid out in matrices, and a 0th column or row is always used for indexing cards, when they are needed.

One last important remark is that we often employ scramble shuffles which are essentially methods of uniformly randomly permuting certain rows or columns of a matrix of cards. Our variants are directly inspired by the double scramble shuffle in [19] which in turn was inspired by the pile scramble shuffle in [8], but similar methods have been used in other physical zero-knowledge proofs in the introduction. Both papers suggest that the employment of envelopes should make such a process physically possible.

2.1.1 Deployment of Cards

The first couple of steps explain how a prover encodes a solution.

**Step 1:** The prover should privately mark every edge in the Hamiltonian cycle with \( E_1(1) \) and every other edge with \( E_1(0) \), i.e. they place a single heart face down on every edge that is part of the cycle and a single spade face down on every edge that is not part of the cycle.

**Step 2:** The prover privately marks every edge with a \( E_n(k) \) where \( k \in 0, 1, \ldots, n \). Physically this means the prover places a face down stack of \( n \) cards encoding some number \( k \) onto each edge of the graph. This is done in the following way: the prover arbitrarily picks one edge in the cycle and marks it with \( E_n(1) \) and marks one of the two edges adjacent to it and also in the cycle with \( E_n(2) \). The prover marks the one edge in the cycle which is unmarked and adjacent to \( E_n(2) \) with \( E_n(3) \). They continue this way, marking adjacent edges of the cycle with consecutive encoded numbers until the last marker \( E_n(n) \) is placed. For an actual Hamiltonian cycle, this marker will necessarily be placed in the last unmarked edge of the cycle, next to \( E_n(1) \). The edges not in the cycle should be marked with \( E_n(0) \).

2.1.2 Checking the Solution

The prover and verifier go through a process of checking every vertex publicly.

**Step 3:** The cards on every edge connected to the vertex currently being checked are used to create a matrix. In column 0 a series of indexing cards ♥1, ♥2, . . . , ♥d are
placed where $d$ is the degree of the vertex being checked. Now for each edge connected to the vertex that is being checked, in column 1 place face down the single cards, $E_1(1)$ or $E_1(0)$, encoding whether that edge is a part of the Hamiltonian cycle or not. In column 2, place face down the $E_n(k)$ markers, i.e. the stack of cards encoding each edge’s number (See Figure 2). Flip the indexing cards face down so that all cards in the matrix are now face down.

**Step 4**: Perform a row scramble shuffle, which is defined by randomly permuting the $d$ rows of the matrix. Then flip over all cards in column 1. If there are not 2 ♥’s in the column, the verifier rejects the solution outright and the protocol terminates. If there are 2 ♥’s in the column, proceed.

**Step 5**: Call the two rows with hearts row $i$ and row $j$. Take the stacks in column 2 of rows $i$ and $j$ and use them to create a second matrix of cards as follows. Column 0 should be a column of face down indexing cards 1, 2, ..., $n$. Column 1 is the stack of cards from the second column of the $i^{th}$ row of the first matrix. These cards should be laid out face down, one by one, preserving the order meaning that a card in the $k^{th}$ spot would be placed in the same row as the $k^{th}$ indexing card. Column 2 is filled the same way but with the stack from the second column and $j^{th}$ row of the first matrix.

**Step 6**: Shift every card of column 2 of the second matrix down one row with the $n^{th}$ card cycling back to the first row. (See Figure 3)

**Step 7**: Perform a row scramble shuffle. Flip over all cards in column 1 and 2. If there are 2 hearts in the same row then the verifier should accept this vertex. Either way, turn all cards face down, perform another row scramble shuffle, then turn over the indexing cards in column 0. Put the rows in correct order using the indexing cards. Then shift all the cards in column 2 up one row, with the card in the first row going to the $n^{th}$ row.

**Step 8**: If the verifier has accepted the vertex, go to step 9. If the verifier has not accepted the vertex, shift every card in column 2 up by 1 row with the card in the first row going to the last row. Turn the indexing cards face down, and perform a row scramble shuffle on the matrix. Flip over the cards in column 1 and 2. If there are not 2 ♥’s in the same row, the verifier should reject the solution entirely and the protocol terminates. If there are 2 ♥’s, the verifier should accept the vertex. Turn all cards face down, perform the row scramble shuffle, flip over column 0 and use the indexing cards to place the rows in their original places, shift all cards in column 2 down a row, with the card in the last row, going to the first row.

**Step 9**: Reform the stack of cards from which column 1 of the second matrix was made and return this stack to the second column and $i^{th}$ row of the first matrix. Likewise, reform the stack of cards from which second column of the second matrix was made and return it to second column and $j^{th}$ row the first matrix. Now there is only the first matrix. Turn any cards face down and perform a row scramble shuffle. Finally, flip over the indexing cards in column 0 and use them to re-deploy the marking cards onto the edges from which they came.

**Step 10**: Repeat steps 3 through 9 for all vertices on the graph.

**Step 11**: Once the verifier has accepted every vertex, place each of the stacks marking the $E_n(k)$ from every edge of the graph into a single column. The column should have the same length as the number of edges in the graph. Perform a row scramble shuffle. Now reveal all stacks of cards and check that there exists exactly $1$ $E_n(i)$ for all $1 \leq i \leq n$. If this is true then the verifier should accept the solution. Otherwise the verifier should reject it.
2.2 Explaining the Steps

We have split the protocol into steps so as to explain intuitively the reasons behind each step.

Step 1 Reason: The prover needs some way to encode their solutions into the graph, if all the cards were turned face up after this step, one would clearly see the Hamiltonian cycle.

Step 2 Reason: The prover now introduces some seemingly extraneous information to their solution, but this extra information allows the verifier to confirm that the prover has one cycle and not multiple, as we shall see below.

After the first two steps, the prover has privately marked the graph with the solution and the rest of the protocol is done publicly, so this is a good time to introduce the intuition of the protocol.

1. Every vertex must be connected to exactly two edges which are in a cycle.

2. A Hamiltonian cycle must contain exactly n edges which occur sequentially one after the other.

The first condition guarantees that any vertex is apart of a cycle. Two disjoint cycles which cover the graph still satisfy the first condition, but not the second. The protocol is then designed to convey that these first two conditions are met, without revealing the Hamiltonian cycle itself.

Step 3 Reason: The remainder of the steps are done publicly. Cards are mostly placed face down so as not to reveal too much information to the verifier, but the creation of the matrices should be done publicly so that neither the verifier nor prover
Figure 3: This is an example of a second matrix from step 6 in which it is checked that adjacent edges in the Hamiltonian cycle are encoded with adjacent numbers modulo the number of vertices, which in this case is 3. (Note this example does not coincide with the example in Figure 2 since in that case, the second matrix would have 5 rows.)

can tamper with the cards that prover has submitted as a solution.

**Step 4 Reason:** This step checks the first condition of the intuition, that every vertex is connected to exactly two edges in the cycle. The scramble shuffle is employed so that the verifier cannot see which edge’s encoding cards belong to which columns.

**Step 5 Reason:**
The second condition still needs to be checked so the other set of marking cards, the card encoding the edges’ numbers, must be checked. This check is prepared by laying out the two stacks into two different columns of a new matrix. Thus the two columns now encode the two numbers associated with the two edges in the Hamiltonian cycle connected to the checked vertex.

**Step 6 Reason:** The idea behind this step is that the numbers marking adjacent edges should have a difference of 1 modulo \( n \). Therefore there are two possibilities. Either the first column of the second matrix encodes the larger number or the second column does. We start by checking the former, but in order to shuffle the columns and preserve the adjacency of the two numbers, we first attempt to make the numbers equal. If the second column is the smaller number, step 6 makes both columns encode the same number.

**Step 7 Reason:** If, in step 6, it was indeed true that the second column originally encoded the smaller number, the verifier should see that the two columns encode the same number after the second column was shifted down. A row scramble shuffle allows the verifier to check this without the prover revealing the numbers themselves. After this check the cards should be put back in the positions in which they started, in the second matrix.

**Step 8 Reason:** In Step 7 Reason, we assumed the second column encoded the
Figure 4: The graph above is showing the encoding of a non-Hamiltonian cycle that would escape local vertex checks. In the example, the prover has encoded two cycles with alternating $E_8(1)$’s and $E_8(2)$’s. The global check however would reveal that $E_8(1)$ and $E_8(2)$ occur multiple times and no other $E_8(k)$ occurs for $k \neq 0, 1, 2$. Thus the verifier would reject.

2.3 Proofs of Security

Lemma 2.1. Perfect Completeness: If the prover has a Hamiltonian cycle, then the verifier always accepts the solution.

Proof. If the prover provides an actual Hamiltonian Cycle and deploys cards in the manner described in the first two steps, then every vertex has exactly two edges that are marked with a heart, and therefore the verifier approves step 4. If the prover provides an actual Hamiltonian cycle, then the two edges adjacent on any given vertex are encoded by consecutive numbers mod $n$, i.e. the stacks in the second column of rows $i$ and $j$ are $E_n(k)$ and $E_n(k-1)$ or $E_n(k-1)$ and $E_n(k)$. If the first is the case, then the shift down in step 6 puts the hearts for the encoding cards in the same row and the verifier accepts.
and the verifier accepts on step 7. If the latter is the case, then once everything has been put back in its original place, the shift up in step 8 puts the two hearts in the same column and the verifier accepts in step 8.

If the prover has provided an actual Hamiltonian cycle the verifier accepts for every vertex, and the protocol moves on to step 11. If the prover provides a valid Hamiltonian cycle, then there is exactly 1 \( E_n(i) \) for every \( i \in \{1, \ldots, n\} \) and the verifier accepts on step 11 and therefore accepts the whole solution.

**Lemma 2.2.** Perfect Soundness: If the prover does not know a Hamiltonian Cycle, then the verifier always rejects.

Proof. We prove the contrapositive. Assume that the verifier accepts. Pick an arbitrary vertex \( v_1 \). We say that two vertices are path adjacent, if the prover has marked the edge between them with a heart. The verification process confirms that each vertex, is path adjacent to two other vertices, so let \( v_2 \) be path adjacent to \( v_1 \) and in turn \( v_3 \) to \( v_2 \) and so on, so that we may list path adjacencies like

\[
v_1, v_2, v_3, \ldots
\]

Since the graph has a finite number of vertices, this either terminates, meaning we get to a vertex with only one path adjacent vertex, or it goes in a cycle. The first case cannot happen if the verifier has accepted, because a vertex without exactly 2 path adjacencies would not be accepted. Thus this is a cycle provided by the prover. However, we still need to show that every vertex appears in the cycle.

Note that edges must be encoded with numbers in \( \{0, 1, 2, \ldots, n\} \) with numbers in \( \{1, \ldots, n\} \) being used exactly once. The global check, at the end of the verification process, ensures that this is the case. Let us consider the edges from the cycle above as

\[
e_{1,2}, e_{2,3}, \ldots
\]

The verification process also checks that these edges are encoded with consecutive numbers modulo \( n \). Thus this cycle cannot have less than \( n \) edges because that would require the prover to use a number in \( \{1, \ldots, n\} \) multiple times or for the some edges to not be encoded with consecutive numbers. Therefore the cycle indeed contains all the vertices.

**Lemma 2.3.** Zero-knowledge: During the verification process, the verifier learns nothing about the prover’s solution.

Proof. The backs of the cards are identical, so the only time that the verifier can learn any information about the solution is when the cards are face side up. Therefore the only steps in which the verifier could potentially get information about the solution are 4, 7, 8 and 11.

The verifier cannot get information at step 4 because of the shuffle in step 3. The probability of the two hearts landing in a pair of rows is uniformly distributed, and thus, from the verifier’s perspective, all pairs of edges are equally likely to be the edges of the Hamiltonian cycle.

Similarly, the shuffles prevent the verifier from getting information at steps 7, 8, or 11. The shuffles hide the number with which each edge is encoded by making all numbers uniformly probable.
Finally note that index cards are only turned face up after all cards have been turned face down and shuffled, thus preventing the verifier gleaning any information from them.

(If you are trying this with an actual deck of cards, the added information of the value on the marking cards as opposed to suit does leak information to the verifier, but if you are dealing with a stripped down system where the suit is only information that is conveyed, say by making every marking card an ace, then no information is leaked.)

3 Flow Free

The rules of Flow Free on are simple. To explain, we call each square in a Flow Free grid a cell. We call the edges between the cells walls. In a rectangular grid, all cells besides the ones at the corner have four walls, and each wall is the wall of two different cells. Two cells are adjacent to one another if they share a wall. Two walls are adjacent to one another if they are walls of the same cell. There is a special type of cell, called a terminal cell. Terminal cells are given at the outset of the game and come in pairs. The pairs are labeled with colors or numbers and the player’s goal is to connect the cells within all pairs via a path. A path contains walls and cells through which it passes. In a solution, walls and cells are only every contained in at most one path. Two walls are path adjacent if they are adjacent and contained in the same path. A solution path for a pair of terminal cells must thus pass through adjacent walls starting at the wall of one of the terminal cells and ending at the wall of the other. Finally every cell must be contained in exactly one solution path; see Figure 1.

3.1 The Protocol

3.1.1 The Prover’s Card Deployment

Let $k$ be the number of pairs of given terminal cells in an $m \times n$ grid. First the prover should privately place stacks of cards on each wall in the manner described below. Once the solution is encoded, there will be two stacks of cards on each wall, so call this first stack, type I. Type I stacks encode the $E_k(j)$. $E_k(j)$ should be placed on every wall through which the path between the $j^{th}$ pair of terminal cells passes, and if no path passes through a given wall then the prover should place $E_k(0)$. The prover then needs to privately place a second stack of cards on each wall. These are type II stacks, which encode the $E_{(n \times m) - k}(i)$. For walls that are not in a solution path, the prover places $E_{(n \times m) - k}(0)$. The prover should also pick a wall of a terminal cell which is in a path, onto which, they place a second stack $E_{(n \times m) - k}(1)$. On the wall that is path adjacent to this one, the prover should place $E_{(n \times m) - k}(2)$, and so on until the path path terminates. On the final wall in this path, which in a correct solution is the wall of the another terminal cell, the prover places $E_{(n \times m) - k}(i)$, where $i$ would be the number of walls contained in the path. For the next terminal cell wall of a path not yet encoded by a type II stack of cards, the prover performs the same steps as were done on the previous path but starts by encoding the first wall with $E_{(n \times m) - k}(i + 1)$. The second wall should be encoded with $E_{(n \times m) - k}(i + 2)$ and so on. This process is repeated until all walls are encoded by two stacks of cards. We note that the prover can do this by putting down the stacks of cards in any order so as not to accidentally give away anything by doing the steps above in the order they are written i.e. this is done privately. See Figure 5.
Figure 5: This is an example of a board with cards deployed. On each wall, two stacks of cards have been placed. In this cases, the smaller stack has only two cards, since $k = 2$. The larger stack has seven cards which comes from $m \times n - k = 3 \times 3 - 2$. The star in the center indicates the cell currently being checked.

### 3.1.2 Checking the Terminal Cells

First we must check the terminal cells publicly. It is already public knowledge that this cell corresponds to the $i^{th}$ path.

**Step 1**: Take the type I stacks from all walls that are adjacent to a given terminal cell and use them to construct a matrix. Suppose it is a terminal cell from the $i^{th}$ pair. Each of the rows is constructed by placing the cards from each of the stacks face down and maintaining the order of the stack meaning a card in the $i^{th}$ position of the stack ends up in the $i^{th}$ column of the matrix. Each stack should have its own row and indexing cards should be placed in the 0th column to keep track of which row corresponds to which wall of the terminal cell.

**Step 2**: Turn the indexing cards face down and perform a row scramble shuffle.

**Step 3**: Turn over the the cards in columns 1 through $k$. If the verifier sees exactly 1♥ in the $i^{th}$ column and no ♥’s anywhere else, the verifier accepts this cell. Otherwise, the verifier rejects the whole solution.

**Step 4**: Turn all cards face down perform a row scramble shuffle, flip over the indexing cards and use them to redeploy the stacks of cards on the board.

**Step 5**: The verifier repeats the process for all terminal cells. If the verifier accepts for all terminal cells, then move on to checking the non-terminal cells.

### 3.1.3 Checking the Non-terminal Cells

Next the non-terminal cells are checked.

**Step 1**: Pick a non-terminal cell. Construct a matrix such that row 0 and column 0 are indexing cards. Now take the type I stack from one of the adjacent walls, placing the cards from this stack face down to form row 1. As usual ordering should be maintained so that a card in the $i^{th}$ position in the stack ends up in the $i^{th}$ column. Then place the entire type II stack in its own entry in the first row at the end, in column $k + 1$. Repeat this process for each adjacent cell wall, adding a new row to the matrix each
time. Let \( r \) be the number of walls associated with this cell. The largest row index is thus \( r \). See \ref{fig:flow_free_puzzle}

**Step 2:** Turn all indexing cards face down and randomly permute rows 1 through \( r \). Then randomly permute the columns 1 through \( k \).

**Step 3:** Flip over all the cards that are in both columns 1 through \( k \) and rows 1 through \( r \). If there are exactly 2 \( \heartsuit \)'s in the same column and no other \( \heartsuit \)'s, the verifier should accept and move to the next step. Otherwise the verifier should reject the solution.

**Step 4:** If the verifier accepts the previous step, then take the type II stacks in the \( k + 1 \) column from the two rows containing the hearts and create a second matrix by deploying them in two columns in the same way it was done in step 5 of the Hamiltonian cycle protocol, but note, the number of rows in the new matrix is \( n \times m - k \) instead of \( n \).

**Step 5:** Perform steps 6-8 of the cycle protocol including the relevant acceptances or rejections.

**Step 6:** If the solution is not rejected in step 5, turn the columns of the second matrix back into the original stacks and put the stacks back into the first matrix, just as was done in the cycle checking protocol.

**Step 7:** Turn all cards face down and randomly permute rows 1, 2, \ldots, \( r \) and then the columns 1, 2, \ldots, \( k \).

**Step 8:** Flip over the indexing cards in row 0 and use this to put the columns back in the right order.

**Step 9:** Flip over the indexing cards in column 0 and use this to redeploy the stacks of cards onto the correct walls as they originally were.

**Step 10:** Repeat the process for every non-terminal cell. If it is accepted for every cell, move to checking the global properties.

### 3.1.4 Checking global properties

A final global check is needed.

**Step 1:** Take the type II stacks from every wall in the puzzle. Line them up face down so that each stack is laid out in a column and perform a column scramble shuffle.

**Step 2:** Turn all cards face up and check that there is exactly 1 heart in each row. If there is, then the verifier should accept the solution to the whole puzzle. If not, the verifier should reject the solution.

### 3.2 Proof of Security

**Lemma 3.1.** Perfect Completeness: If the prover has a solution to a Flow Free Puzzle, the verifier always accepts the solution.

**Proof.** If the prover provides a card deployment that corresponds to a solution, then there is exactly 1 \( \heartsuit \) in the right column in step 3 of each terminal cell check. Therefore the verifier accepts the terminal cells and moves on to the non-terminal cells.

Each non-terminal cell must have 2 adjacent walls with stacks of type I that have \( \heartsuit \)'s in the same column at deployment, while the other walls should have stacks of type I with no heart. These facts are not changed by either the random permutation of the rows or columns, therefore the verifier accepts step 3 of the non-terminal cell check. The two stacks of type II, corresponding to the walls with the stacks of type I with
Figure 6: This is the first matrix of cards that should be formed in checking the non-terminal cell marked with a star in Figure 5. The type I stacks from the four adjacent walls have been laid out so that the $i^{th}$ card in the stack from $j^{th}$ wall is in the $(i,j)$ position. The last column contains the type II stacks. (There appear to be only five cards in each of the type II stacks, but in the example there would be $3 \times 3 - 2 = 7$.)
hearts, should encode adjacent numbers i.e. \( E_{nxm-k}(j) \) and \( E_{nxm-k}(j + 1) \) therefore the verifier accepts during step 5 for all non-terminal cells.

Checking the global properties, the verifier accepts in step 2 because the prover has deployed one stack \( E_{nxm-k}(j) \) for each \( j \in \{1, \ldots n \times m - k\} \), since this is exactly the number of walls that are occupied by a path, in a valid solution. Therefore if presented with a valid solution the verifier always accepts.

\[\Box\]

**Lemma 3.2.** Perfect Soundness: If the prover does not know a solution to a Flow Free Puzzle, then the verifier always rejects.

**Proof.** We prove the contrapositive. Assume that the verifier accepts. A proper solution has two main objectives: connect each pair of terminal points via a path and fill the grid so that each cell is contained in exactly one path which terminates at terminal cells.

We say a terminal cell belongs to a path if it is one of the terminal cells of the \( i \)th pair, and it has exactly one wall in a path, and that wall is in path \( i \). The check on terminal cells confirms that all terminal cells belong to the correct path. We say that a non-terminal cell belongs to a path \( i \) if it has exactly two walls in a path \( i \), and its other walls are not in a path. The check on non-terminal cells, confirms that each non-terminal cell belongs to a possibly non-terminating path. These facts guarantee that all paths that do terminate, do so at terminal cells. If a path terminates at a non-terminal cell, the non-terminal cell would not get past the verification process. Furthermore, each terminal cell is in a path that terminates at another terminal cell in the same path.

If a path does not terminate, it must be a cycle and it must not contain terminal cells. Therefore any cycle has strictly less than \( n \times m - k \) walls. However the verification process checks that path adjacent walls are encoded with consecutive numbers modulo \( n \times m - k \), so such a cycle must repeat a number e.g. 1, 2, 3, 2, ..., but the global check would reject repeated numbers, so there must not be any cycles. Thus the prover does indeed have a valid solution, since this shows that all non-terminal cells belong to paths that terminate at the correct terminal cells.

(Note, the underlying graph structure (walls = edges, cells = vertexes) of any Flow Free game is simple graph so it is impossible to have a 2 cycle.)

\[\Box\]

**Lemma 3.3.** Zero-knowledge: During the verification process, the verifier learns nothing about the prover’s solution.

**Proof.** The only times that the verifier can learn about prover’s solution is when the cards are flipped face up, i.e. at steps 3 and 4 of the terminal cell check, steps 3, 5, 8, and 9 of the non-terminal cell check and step 2 of the global check. The revealing of indexing cards never gives information because a random permutation with all cards face down is always performed before hand.

Step 3 of the terminal cell check gives no retrievable information about which wall is used because of the random permutation during step 2. The information about which path it corresponds to is retrievable, but that is already public information so no information is given.

Step 3 of the non-terminal cells gives no retrievable information because of the random permutations in step 2. Step 5 gives no retrievable information for the reasons provided in the Hamiltonian graph section. Steps 8 and 9 give no retrievable information because the random permutations in step 7 strips the information provided in step 3.
Step 2 of the global check gives no information because the random permutation in step 1 strips the information of which stack corresponds to which wall.

4 Many-to-Many k-Disjoint Path Cover and Conclusion

The many-to-many k-disjoint path cover problem studied in [10], [15], [16], [17], [23] looks at breaking a graph into k disjoint paths, with given starting and ending vertices (sources and sinks) such that every vertex is in exactly one path. We say it is paired if each source is required to be in the same path as a specified sink, otherwise it is unpaired. Finding a Hamiltonian path with given starting points is a special case of the many-to-many k-disjoint path cover problem, where k = 1. Flow Free puzzles are also special cases of the problem if we look at the underlying adjacency graph of the board. If we simply replace cell with vertex and wall with edge, our protocol generalizes to providing a zero-knowledge proof for the paired many-to-many k-disjoint covering path problem for simple graphs and we can also handle the unpaired many-to-many k-disjoint covering path problem by including a column scramble when checking all of the sink vertices.

The slight variation for Flow Free rules meant that the protocol described in [19] could not be neatly extended and so substantial changes were needed as well as an introduction of seemingly extraneous information. This mean that our protocol required orders of magnitude more cards. It uses $\Theta((mn)^k)$ cards on an $m \times n$ grid with $k$ pairs of terminal cells. The $k$ does not appear since although a stack of $k$ cards is needed on each wall, a stack of $m \times n - k$ is needed also needed on each wall. Generalizing this to graphs we would have $\Theta(|E||V|)$ cards for both the Hamiltonian cycle and many-to-many k-disjoint covering paths. A fruitful direction for future research may be seeing if there exists a method that can achieve the same perfect soundness while using fewer cards. Furthermore, our checking algorithm requires many steps, and an algorithm that can check more efficiently would be another possible direction for improvement. Finally, we encourage generalizations of these to protocols to directed graphs, other games, logic puzzles, and other problems in algorithmic graph theory.

References

[1] Aaron Adcock et al. “Zig-Zag Numberlink is NP-Complete”. In: Journal of information processing (Tokyo) 23.3 (2015), pp. 239–245. DOI: [10.2197/ipsjjip.23.239].

[2] Manuel Blum. “How to prove a theorem so no one else can claim it”. In: Proceedings of the International Congress of Mathematicians. Vol. 1. 1986, pp. 1444–1451.

[3] Uriel Feige, Dror Lapidot, and Adi Shamir. “Multiple NonInteractive Zero Knowledge Proofs Under General Assumptions”. In: SIAM Journal on Computing 29 (1999), pp. 1–28. DOI: [10.1137/S00975397972230010].

[4] Flow free - apps on Google Play. URL: [https://play.google.com/store/apps/details?id=com.bigduckgames.flow&hl=en_US&amp;gl=US](https://play.google.com/store/apps/details?id=com.bigduckgames.flow&hl=en_US&amp;gl=US).
[5] Flow free: Bridges - apps on Google Play. URL: https://play.google.com/store/apps/details?id=com.bigduckgames.flowbridges&hl=en_US&gl=US.

[6] Flow free: Hexes - apps on Google Play. URL: https://play.google.com/store/apps/details?id=com.bigduckgames.flowhexes&hl=en_US&gl=US.

[7] Ronen Gradwohl et al. “Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles”. In: Fun with Algorithms. Ed. by Pierluigi Crescenzi, Giuseppe Prencipe, and Geppino Pucci. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 166–182. ISBN: 978-3-540-72914-3.

[8] Rie Ishikawa, Eikoh Chida, and Takaaki Mizuki. “Efficient Card-Based Protocols for Generating a Hidden Random Permutation Without Fixed Points”. In: Unconventional Computation and Natural Computation: 14th International Conference, UCNC 2015, Auckland, New Zealand, August 30 – September 3, 2015, Proceedings. Vol. 1. 2015, pp. 215–226. DOI: 10.1007/978-3-319-21819-9.

[9] Mohammad Sina Kiarostami et al. “Multi-Agent non-Overlapping Pathfinding with Monte-Carlo Tree Search”. In: 2019 IEEE Conference on Games (CoG). 2019, pp. 1–4. DOI: 10.1109/CIG.2019.8848043.

[10] Brian G. Kronenthal and Wing Hong Tony Wong. “Paired many-to-many disjoint path covers of hypertori”. In: Discrete Applied Mathematics 218 (2017), pp. 14–20. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2016.09.020 URL: https://www.sciencedirect.com/science/article/pii/S0166218X16304127.

[11] Daiki Miyahara et al. “Card-Based Physical Zero-Knowledge Proof for Kakuro”. In: IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E102.A.9 (2019), pp. 1072–1078. DOI: 10.1587/transfun.E102.A.1072.

[12] Daiki Miyahara et al. “Card-Based ZKP Protocols for Takuzu and Ju- osan”. In: 10th International Conference on Fun with Algorithms (FUN 2021). Vol. 157. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 20:1–20:21. DOI: 10.4230/LIPIcs.FUN.2021.20 URL: https://drops.dagstuhl.de/opus/volltexte/2020/12781.

[13] Shlomo Moran and Yaron Wolfstahl. “Optimal covering of cacti by vertex-disjoint paths”. In: Theoretical Computer Science 84.2 (1991), pp. 179–197. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(91)90159-4 URL: https://www.sciencedirect.com/science/article/pii/030439759190159Y.

[14] Tobias Nef et al. “Development and evaluation of Maze-Like puzzle games to assess cognitive and motor function in aging and neurodegenerative diseases”. In: Frontiers in aging neuroscience (2020), p. 87.
[15] JH Park, HC Kim, and HS Lim. “Many-to-many disjoint path covers in hypercube-like interconnection networks with faulty elements”. eng. In: *IEEE transactions on parallel and distributed systems* 17.3 (2006), pp. 227–240. issn: 1045-9219.

[16] Jung-Heum Park and Insung Ihm. “Many-to-many two-disjoint path covers in cylindrical and toroidal grids”. In: *Discrete Applied Mathematics* 185 (2015), pp. 168–191. issn: 0166-218X. DOI: [https://doi.org/10.1016/j.dam.2014.12.008](https://doi.org/10.1016/j.dam.2014.12.008) URL: [https://www.sciencedirect.com/science/article/pii/S0166218X14005381](https://www.sciencedirect.com/science/article/pii/S0166218X14005381).

[17] Jungheum Park. “Torus-like graphs and their paired many-to-many disjoint path covers”. In: *Discret. Appl. Math.* 289 (2021), pp. 64–77.

[18] Léo Robert et al. “Interactive Physical ZKP for Connectivity: Applications to Nurikabe and Hitori”. In: *Connecting with Computability*. Ed. by Liesbeth De Mol et al. Cham: Springer International Publishing, 2021, pp. 373–384. isbn: 978-3-030-80049-9.

[19] Suthee Ruangwises and Toshiya Itoh. “Physical Zero-Knowledge Proof for Numberlink Puzzle and k Vertex-Disjoint Paths Problem”. In: *New Generation Computing* (2021) 39.1 (2020), pp. 3–17. DOI: [10.1007/s00354-020-00114-y](https://doi.org/10.1007/s00354-020-00114-y).

[20] Suthee Ruangwises and Toshiya Itoh. “Physical zero-knowledge proof for Ripple Effect”. In: *Theoretical Computer Science* 895 (2021), pp. 115–123. issn: 0304-3975. DOI: [https://doi.org/10.1016/j.tcs.2021.09.034](https://doi.org/10.1016/j.tcs.2021.09.034) URL: [https://www.sciencedirect.com/science/article/pii/S0304397521005557](https://www.sciencedirect.com/science/article/pii/S0304397521005557).

[21] Tatsuya Sasaki et al. “Efficient card-based zero-knowledge proof for Sudoku”. In: *Theoretical Computer Science* 839 (2020), pp. 135–142. issn: 0304-3975. DOI: [https://doi.org/10.1016/j.tcs.2020.05.036](https://doi.org/10.1016/j.tcs.2020.05.036) URL: [https://www.sciencedirect.com/science/article/pii/S0304397520303200](https://www.sciencedirect.com/science/article/pii/S0304397520303200).

[22] Ryo Yoshinaka et al. “Finding All Solutions and Instances of Numberlink and Slitherlink by ZDDs”. In: *Algorithms* 5.2 (2012), pp. 176–213. issn: 1999-4893. DOI: [10.3390/a5020176](https://doi.org/10.3390/a5020176) URL: [https://www.mdpi.com/1999-4893/5/2/176](https://www.mdpi.com/1999-4893/5/2/176).

[23] Shurong Zhang and Shiyiing Wang. “Many-to-many disjoint path covers in k-ary n-cubes”. In: *Theoretical Computer Science* 491 (2013), pp. 103–118. issn: 0304-3975. DOI: [https://doi.org/10.1016/j.tcs.2013.04.002](https://doi.org/10.1016/j.tcs.2013.04.002) URL: [https://www.sciencedirect.com/science/article/pii/S0304397513002648](https://www.sciencedirect.com/science/article/pii/S0304397513002648).